(4)

Technical Report 1031

Automatic Qualitative Analysis of Ordinary Differential Equations Using Piecewise Linear Approximations

Elisha Sacks

MIT Artificial Intelligence Laboratory

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AI-TR 1031	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Automatic Qualitative Analysis of Ordinary Differential Equations Using Piecewise Linear Approximations		5. TYPE OF REPORT & PERIOD COVERED technical report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Elisha Peretz Sacks		8. CONTRACT OR GRANT NUMBER(s) N00014-85-K-0124 N00014-86-K-0180
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, MA 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE March 1988
		13. NUMBER OF PAGES 98
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Arlington, VA 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Distribution is unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES None		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) qualitative reasoning dynamic systems qualitative physics symbolic mathematics		

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis explores automating the qualitative analysis of physical systems. Scientists and engineers model many physical systems with ordinary differential equations. They deduce the behavior of the systems by analyzing the equations. Most realistic models are nonlinear, hence difficult or impossible to solve explicitly. Analysts must resort to approximations or to sophisticated mathematical techniques. I describe a program, called PLR (for Piecewise Linear Reasoner), that formalizes an analysis strategy employed by experts. PLR takes

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 I

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20 cont.

parameterized ordinary differential equations as input and produces a qualitative description of the solutions for all initial values. It approximates intractable nonlinear systems with piecewise linear ones, analyzes the approximations, and draws conclusions about the original systems. It chooses approximations that are accurate enough to reproduce the essential properties of their nonlinear prototypes, yet simple enough to be analyzed completely and efficiently.

PLR uses the standard phase space representation. It builds a composite phase diagram for a piecewise linear system by pasting together the local phase diagrams of its linear regions. It employs a combination of geometric and algebraic reasoning to determine whether the trajectories in each linear region cross into adjoining regions and summarizes the results in a ***transition graph.*** Transition graphs explicitly express many qualitative properties of systems. PLR derives additional properties, such as boundedness or periodicity, by theoretical methods. PLR's analysis depends on abstract properties of systems rather than on specific numeric values. This makes its conclusions more robust and enables it to handle parameterized equations transparently. I demonstrate PLR on several common nonlinear systems and on published examples from mechanical engineering.

Automatic Qualitative Analysis of Ordinary Differential Equations Using Piecewise Linear Approximations

Elisha Peretz Sacks

March 24, 1988

This research was supported (in part) by National Institutes of Health Grant No. R01 LM04493 from the National Library of Medicine and National Institutes of Health Grant No. R24 RR01320 from the Division of Research Resources.
This report describes research done in part at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the United States Department of Defense under ONR contract N00014-85-K-0124.

Also appears as LCS-TR-416

Automatic Qualitative Analysis of Ordinary Differential Equations Using Piecewise Linear Approximations

by

Elisha Peretz Sacks

Submitted to the Department of Electrical Engineering and Computer Science
on February 17, 1988
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Abstract

[illegible]

This thesis explores automating the qualitative analysis of physical systems. Scientists and engineers model many physical systems with ordinary differential equations. They deduce the behavior of the systems by analyzing the equations. Most realistic models are nonlinear, hence difficult or impossible to solve explicitly. Analysts must resort to approximations or to sophisticated mathematical techniques. I describe a program, called PLR (for Piecewise Linear Reasoner), that formalizes an analysis strategy employed by experts. PLR takes parameterized ordinary differential equations as input and produces a qualitative description of the solutions for all initial values. It approximates intractable nonlinear systems with piecewise linear ones, analyzes the approximations, and draws conclusions about the original systems. It chooses approximations that are accurate enough to reproduce the essential properties of their nonlinear prototypes, yet simple enough to be analyzed completely and efficiently. [illegible]

PLR uses the standard phase space representation. It builds a composite phase diagram for a piecewise linear system by pasting together the local phase diagrams of its linear regions. It employs a combination of geometric and algebraic reasoning to determine whether the trajectories in each linear region cross into adjoining regions and summarizes the results in a *transition graph.* Transition graphs explicitly express many qualitative properties of systems. PLR derives additional properties, such as boundedness or periodicity, by theoretical methods. PLR's analysis depends on abstract properties of systems rather than on specific numeric values. This makes its conclusions more robust and enables it to handle parameterized equations transparently. I demonstrate PLR on several common nonlinear systems and on published examples from mechanical engineering.

Thesis Jointly Supervised by:

Gerald Jay Sussman	Professor, Electrical Engineering
Ramesh Patil	Assistant Professor, Electrical Engineering and Computer Science

Acknowledgements

I would like to thank the following people for their help during my graduate student years.

Ramesh Patil for good advice, for teaching me engineering and AI, for supervising my research, and for proof-reading this thesis several times (even while in India).

Gerry Sussman for broadening my horizons, deepening my understanding, stimulating my imagination, and clarifying my presentation.

Peter Szolovits for good advice, for generous financial support, and for teaching me AI and physics.

Mike Wellman for his friendship, for thoughtfully discussing my research and many other topics, and for carefully proof-reading this thesis and many other papers. Your knowledge and insight have enhanced my work and clarified my writing.

My office-mates Dennis Fogg, Ira Haimowitz, and Alex Yeh and my longtime friends Phyllis Koton and Tom Russ for years of friendship, fun, ideas, and support.

My friends and colleagues in the clinical decision-making group and throughout M.I.T. for making this a rich, stimulating research environment.

My wife Jennifer for her constant love, support, and supply of peanut-butter sandwiches.

This research was supported (in part) by National Institutes of Health Grant No. R01 LM04493 from the National Library of Medicine and National Institutes of Health Grant No. R24 RR01320 from the Division of Research Resources.

Contents

List of Figures

Chapter 1

Introduction

Differential equations provide a rich language fo⁻ describing smoothly varying phenomena. Scientists and engineers model many physical systems with differential equations, including circuits, motors, buildings, chemical reactions, physiology, the motions of celestial bodies, and the interactions of competing populations. They predict the behavior of the physical systems by deducing the properties of the solutions to the differential equations. This thesis describes a computer program, called PLR (for Piecewise Linear Reasoner), that automates the analysis. It takes parameterized ordinary differential equations as input and produces a qualitative description of the solutions for all initial values.

Although simple mathematical techniques exist for systematically determining the solutions to linear equations, most nonlinear equations require sophisticated idiosyncratic techniques or resist analysis altogether. Artificial intelligence researchers have responded to the apparently refractory nature of the general problem by lowering their level of aspiration to much weaker deductions about the qualitative behavior of physical systems. They have replaced differential equations with impoverished modeling languages and inference methods. I see no need to sacrifice the powerful tools developed by generations of mathematicians just because they do not solve *every* differential equation directly. Instead, I have developed a program that emulates a strategy whereby human experts successfully apply those tools to complicated equations.

Experts transform intractable equations into tractable ones by principled approximations that preserve the significant properties of the original equations. The approximations also offer a framework for more detailed investigation if necessary. Experts try linear approximations first. When these prove inadequate, they shift to piecewise linear ones. Piecewise linear approximations can be made accurate enough to reproduce the essential properties of their nonlinear prototypes, yet simple enough to be analyzed completely and efficiently. The principal goal of this thesis is to demonstrate a mechanical technique for constructing and analyzing piecewise linear approximations of differential equations. The analysis depends on abstract properties of the approximation, such as whether certain lines slope up or down, rather than on specific numeric values. This makes the conclusions more robust and permits the technique to handle parameterized equations transparently.

1.1 Phase Space

PLR uses the standard phase space representation for systems of first-order differential equations, which stresses the long-term qualitative form of solutions for all initial values, while downplaying exact solutions. It converts higher-order equations to first-order ones by introducing new variables as synonyms for higher derivatives. The phase space for a system

$$x_i' = f_i(x_1, \ldots, x_n);\ i = 1, \ldots, n \tag{1.1}$$

with dependent variables $x_1, \ldots, x_n$ is the Cartesian product of the x_i's domains. Points in phase space represent states of the system. Curves on which equation (1.1) is satisfied, called *trajector. ;,* represent solutions. A *phase diagram* for a system depicts its phase space and trajectories graphically. The topological and geometric properties of trajectories characterize the qualitative behavior of solutions. For example, a point trajectory, called a *fixed point,* indicates a constant solution, whereas a closed curve indicates a periodic solution. These abstract descriptions are generally the best that one can do, since most systems lack closed-form solutions.

1.2 The Algorithm

PLR analyzes systems of first-order differential equations according to the algorithm shown in Figure 1.1. Initially, PLR constructs a simple piecewise linear approximation that contains few linear regions. It refines this model into a progression of increasingly accurate ones by repeatedly splitting the linear regions. It compares the phase diagram of each approximation to that of the previous one. Refinement ends when no new qualitative properties appear. PLR only handles systems of two equations, but the underlying algorithm extends to larger systems, as discussed in the final chapter.

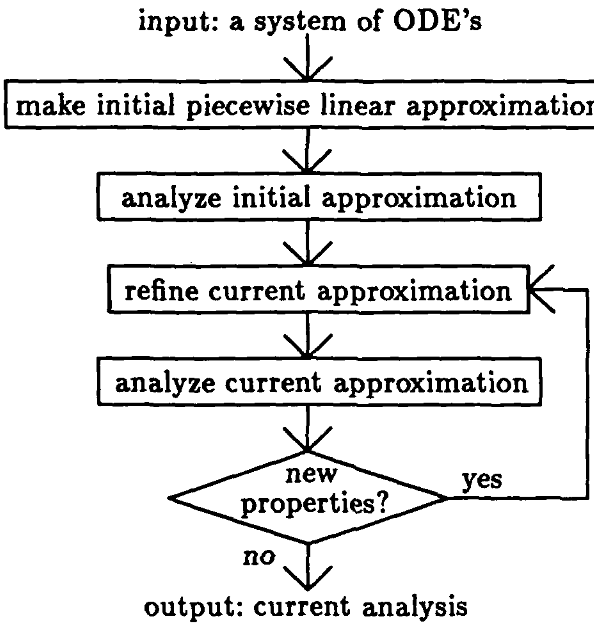

input: a system of ODE's
make initial piecewise linear approximation
analyze initial approximation
refine current approximation
analyze current approximation
new properties?
yes
no
output: current analysis

Figure 1.1: The abstract PLR algorithm

PLR builds a composite phase diagram for a piecewise linear system by combining the local phase diagrams of its linear regions. It employs the standard theory of linear equations to ascertain the local phase diagrams. Linear systems have simple well-understood dynamics. Either all trajectories are periodic, all approach a unique constant solution, called a *fixed point*, or all approach infinity. In some cases, two *asymptotes* partition phase space into four disjoint sets.

PLR pastes together the local phase diagrams by determining which sequences of regions trajectories can traverse. It tests whether trajectories cross between adjacent regions by examining the behavior of the corresponding linear systems on the common boundaries. It summarizes the results in a *transition graph* whose nodes and links represent regions and transitions. Each path through the transition graph of a piecewise linear system indicates that trajectories traverse the corresponding regions in the prescribed order. Transition graphs explicitly express many qualitative properties of systems. Loops denote trajectories that remain in one region forever, whereas longer cycles denote trajectories that continually shift between a sequence of regions. PLR derives additional properties, such as boundedness and periodicity, by global analysis of the original equations.

For example, PLR derives the properties of the equation

$$\theta'' + \frac{g}{l}\sin\theta = 0;\ g, l > 0, \tag{1.2}$$

which describes the angular displacement of an undamped pendulum of length l, by piecewise linearizing the nonlinear term $\sin\theta$, as shown in Figure 1.2. The local phase diagrams of the three linear regions in the (θ, θ') phase plane appear in Figure 1.3. PLR divides the plane into regions A through G (Figure 1.4a), splits D into D1 through D5, and constructs a transition graph (Figure 1.4b). It concludes that trajectories in the cycles C-D2-G-D4 and D1 follow closed paths around the origin, whereas those in F-D5-B and A-D3-E move continually from right to left and from left to right respectively. In physical terms, the pendulum rocks back and forth in the former cases and spins around in the latter. Global analysis of equation (1.2) establishes that every rocking trajectory has a fixed magnitude and period. PLR refines the original linearization by bisecting each interval in the approximation of $\sin\theta$, as shown in Figure 1.5. The new phase diagram (Figure 1.6) fails to produce additional behaviors. PLR assumes that further refinement is pointless and halts.

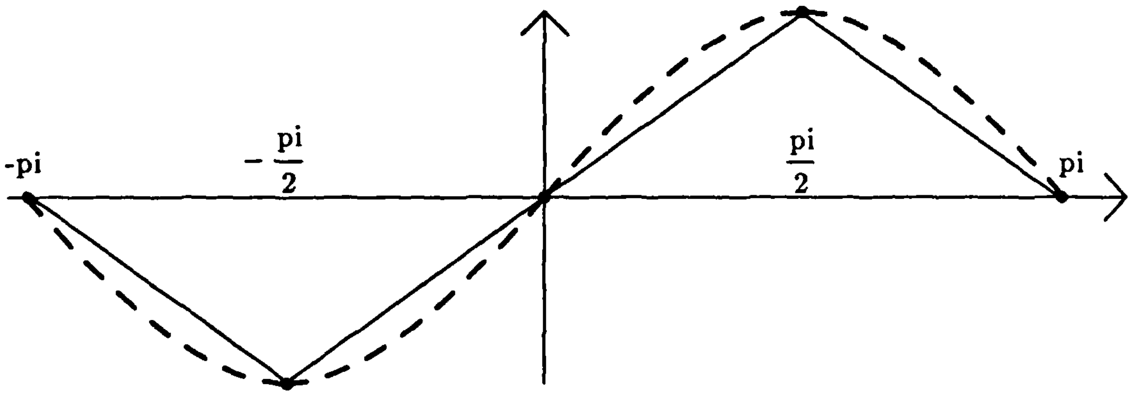


Figure 1.2: Piecewise linearization of $\sin\theta$

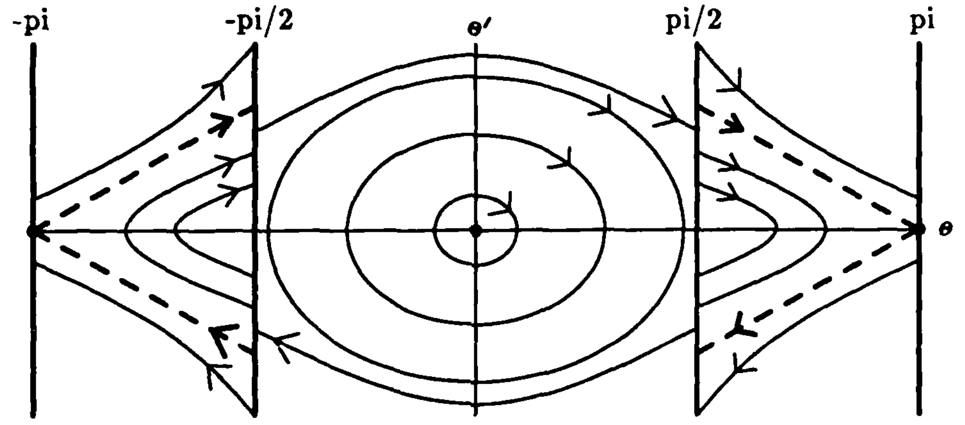


Figure 1.3: Local phase diagrams for the piecewise linear pendulum equations: dots, thick solid lines, and dashed lines indicate fixed points, boundaries between linear regions, and asymptotes.

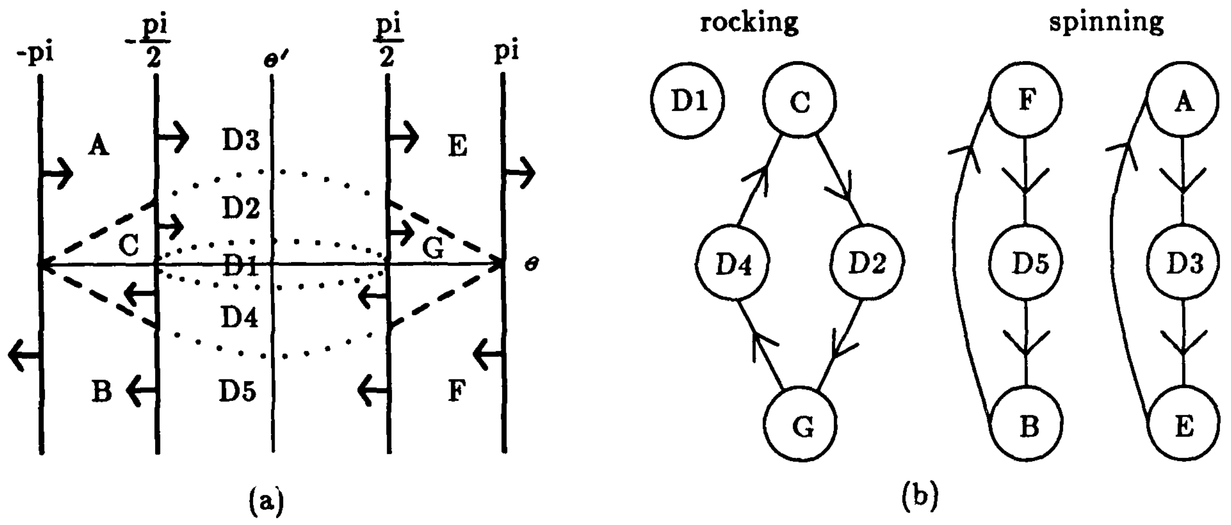


Figure 1.4: (a) Phase diagram and (b) transition graph for the pendulum

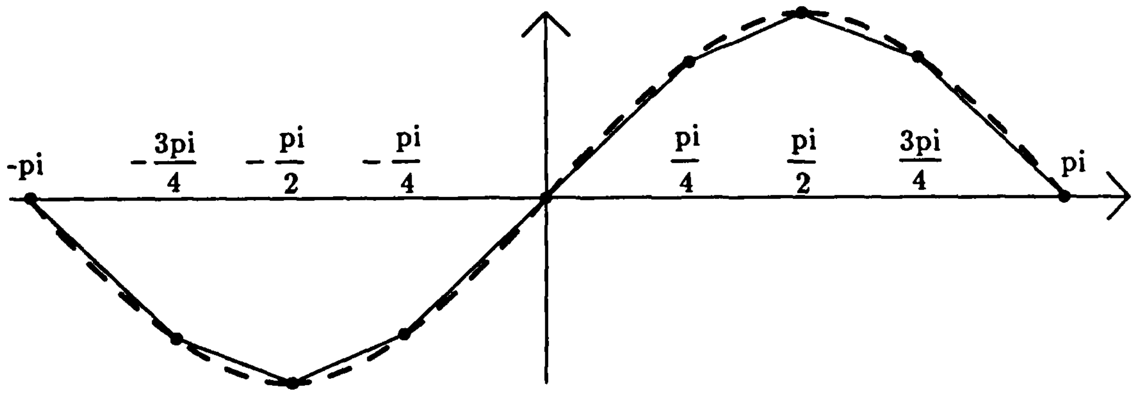


Figure 1.5: Refinement of the piecewise linearization from Figure 1.2 obtained by inserting a new linearization point between each adjacent pair of existing points.

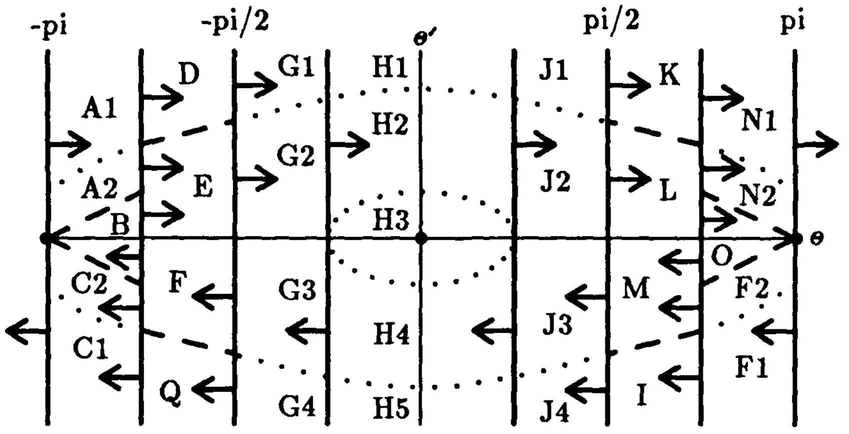


Figure 1.6: Phase diagram for the refined pendulum approximation

1.3 Thesis Overview

The pendulum example illustrates my approach to automatic analysis of differential equations. PLR exploits powerful mathematical techniques, drawn from the repertoire of scientists and engineers, to solve hard problems. It derives the qualitative behavior of complicated equations by constructing and analyzing suitable piecewise linear approximations. It augments the results with direct analysis if applicable. In this thesis, I substantiate my approach by describing PLR in detail and demonstrating its capabilities.

The thesis is organized as follows. I explain how PLR constructs piecewise linear approximations in the next chapter. In the following chapter, I describe the algorithm, called *piecewise analysis,* by which it constructs phase diagrams and transition graphs of piecewise linear systems. In Chapter 4, I demonstrate piecewise analysis on several common nonlinear systems and on published examples from mechanical engineering. Piecewise analysis cannot determi^-^ some global properties of trajectories. For example, it cannot tell that the trajectories of the pendulum have fixed magnitudes. Chapter 5 describes theoretical tools with which PLR deduces global behavior, including that of the pendulum. PLR uses these tools to compare transition graphs and refine piecewise approximations, as explained in chapter 6.

Every stage of PLR requires sophisticated inequality reasoning to determine the qualitative properties of algebraic expressions. Inequalities determine the shape of piecewise linear functions, the behavior of linear systems, the links in transition graphs, and the results of theoretic analysis. In chapter 7, I present an inequality prover that does the job quickly and effectively. It employs a sequence of increasingly sophisticated strategies: when one fails to resolve an inequality, it tries the next. Chapter 8 describes the tools with which PLR manipulates parameterized functions: a mathematical reasoner that derives their extrema, zeros, and other properties and a qualitative sketcher that draws them.

Chapter 9 contains a review of literature and a comparison between PLR and the conventional AI approach involving weak constraints. In the final chapter, I assess PLR's strengths and weaknesses and discuss future work. The biggest challenge is to extend PLR beyond second-order systems. I show that piecewise analysis carries over directly to higher-dimensional systems. To the extent that theoretical techniques for higher-dimensional systems exist, they can automated within the PLR framework.

Chapter 2

Choosing a Piecewise Linear Approximation

The first step in analyzing an unsolvable system of nonlinear differential equations is to approximate it with piecewise linear equations. PLR can construct piecewise linear approximations for many systems automatically. The user need only intervene when it fails, as discussed below. Other analysis systems always require their users to specify approximations. Users of typical numerical packages must choose appropriate algorithms, error margins, initial guesses, and step sizes. Similarly, de Kleer and Brown [7, p. 26] note that qualitative reasoning requires users to derive the confluences for systems by themselves. Like those systems, PLR focuses on analysis, but unlike them, it also has significant model construction capabilities.

The first class for which PLR can make piecewise linear approximations consists of the *separable systems*

$$x_i' = f_i(x_1, \ldots, x_n) = \sum_{j=1}^{n} g_{ij}(x_j); \; i = 1, \ldots, n. \tag{2.1}$$

in which each component $f_i(x_1, \ldots, x_n)$ is a sum of univariate terms. PLR approximates each f_i by replacing its nonlinear terms with piecewise linear ones and adding the results. It chooses a set of *significant points* for each term, fits lines between successive pairs of significant points, and extends the first and last lines to the lower and upper bounds of the phase space. The significant points of a term are its local extrema and finite domain boundaries, components of fixed points of the system, and any additional points designated by the user. The extrema and boundaries guarantee that the approximations will roughly resemble the original terms, whereas the fixed point components align the phase spaces of the approximate and original systems. PLR linearizes terms with a single significant point by bracketing that point with two additional points, as shown in Figure 2.1.

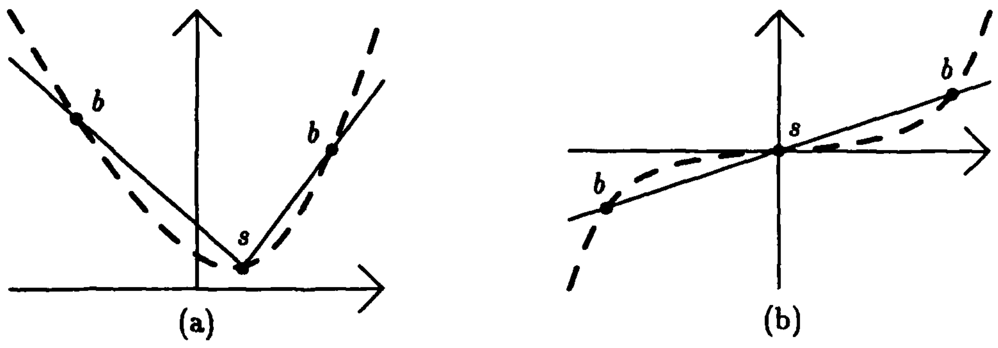


Figure 2.1: Special-case piecewise linearizations (s and b indicate significant and bracketing points): (a) unimodal function without zeroes (b) monotone function

For example, PLR derives the piecewise linear equations

$$\left\{\begin{array}{l} x' = y \\ y' = f(x) - y \end{array}\right. \quad \text{with } f(x) = \left\{\begin{array}{ll} x/2 + 1/2 & \text{for} \quad x \le -1/2 \\ -x/2 & \text{for} \quad x \ge -1/2 \end{array}\right. \tag{2.2}$$

from the Lienard equation

$$x'' + x' + x^2 + x = 0 \tag{2.3}$$

by replacing the nonlinear term $x^2 + x$ with two lines (Figure 2.2). The significant points -1 and 0 are the x components of the system's fixed points $(-1, 0)$ and $(0, 0)$, whereas the significant point $-1/2$ is a minimum. The first line connects the minimum to the zero at -1 and extends to $-\infty$. The second connects the minimum to the zero at 0 and extends to ∞.

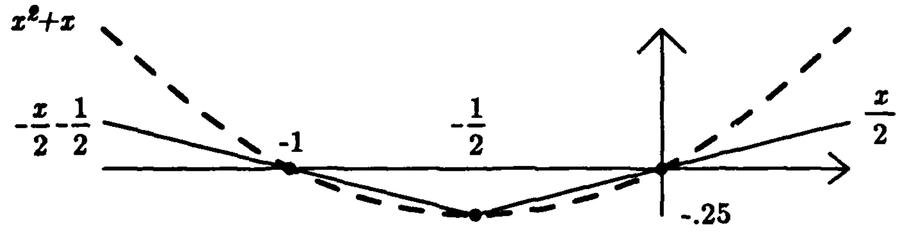


Figure 2.2: Piecewise linearization of $x^2 + x$

The algorithm for separable systems extends to networks of nonlinear components connected by linear laws, such as circuit models. PLR approximates the nonlinear component models with piecewise linear ones as before. The equations for the resulting network are piecewise linear. Users can represent components that display hysteresis with nonplanar phase spaces. Future versions of PLR will automate the process. Figure 2.3 illustrates the phase space for a component that follows the curves 1 through 3 as x increases from $-c$ to c and returns via 4 through 6 as x decreases back to $-c$. Each curve determines a region in phase space. Region 4 lies to the right of 3; region 1 lies to the left of 6. All other adjacencies are planar. I explain how PLR implements nonplanar phase spaces in Section 3.7.

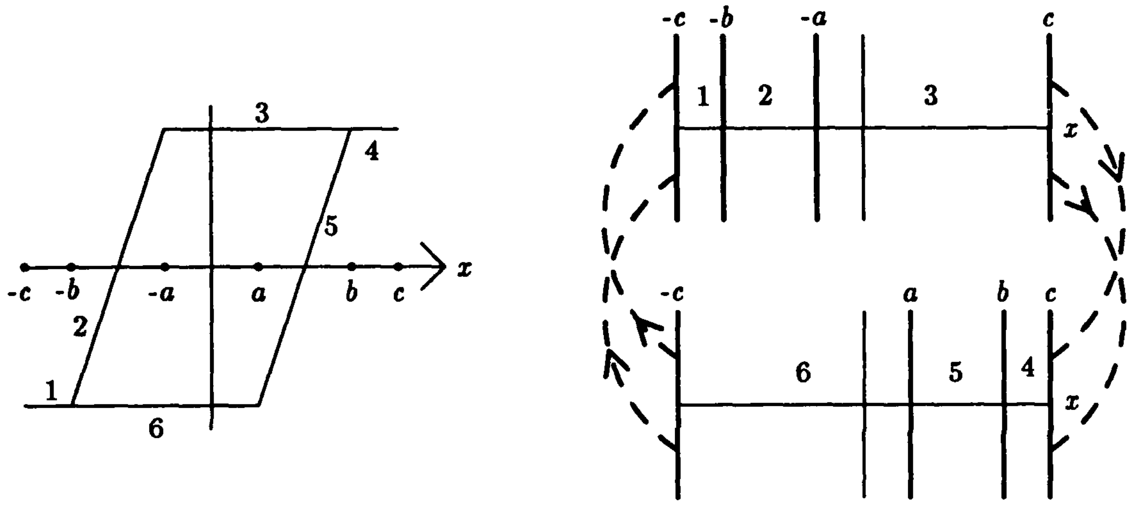


Figure 2.3: Nonplanar phase space for a hysteresis curve

In general, PLR cannot piecewise linearize nonseparable systems, such as $x''(x')^2 = 0$. The previous strategy does not apply to products (and other functions) of piecewise linear functions that contain nonlinear terms. However, PLR handles the important special case

$$x'' + g(x)x' + u(x') + v(x) = 0 \tag{2.4}$$

by recasting it as

$$x'' + G'(x) + u(x') + v(x) = 0 \tag{2.5}$$

with $G(x) = \int g(x)$. It piecewise linearizes G, u, and v according to the algorithm above, substitutes the linearizations into equation (2.5), and converts the resulting piecewise linear second-order equation to a first-order system. An example of this method appears in Section 4.1.

PLR invokes the QMR mathematical reasoner, described in Section 8.1, to calculate the extrema, zeros, and fixed points of nonlinear univariate functions. When QMR fails, the user must n this
thesis except for the function $g(\theta) = f(\theta + a) + f(\theta - a)$ with

$$f(\theta) = (r^2 + 2r + 2 - 2r\cos\theta)^{-3/2}\sin\theta \tag{4.10}$$

and with r and a positive constants, which describes the force applied to a pendulum by a horseshoe magnet (Sec. 4.3). QMR lacks the prerequisite trigonometric manipulation skills for calculating the extrema of g. A better symbolic algebra system could handle functions like g, but no system can analyze every useful function [36].

Piecewise linearization involves a tradeoff between accuracy and simplicity: more lines produce more accurate results at the price of larger transition graphs and phase diagrams. PLR resolves the conflict by constructing simple initial approximations, which only capture the most basic features of the original equations, and refining them when necessary. In thc next two chapters, I will demonstrate that these initial approximations often provide accurate pictures of qualitative behavior. I will discuss how and when PLR refines them in Chapter 6.

Chapter 3

Constructing a Phase Diagram for a Piecewise Linear System

PLR constructs a phase diagram for a second-order system of piecewise linear equations by partitioning phase space into regions in which the system is linear, analyzing the linear systems, and combining the results. It pastes together the local analyses into a global phase diagram by determining which sequences of regions trajectories can traverse. It does this in two stages: transition analysis and case analysis. Define region R to *access* region S if and only if some local trajectory of R crosses into S. In transition analysis, PLR constructs the accessibility relation between pairs of adjacent regions. In case analysis, it extends the results to sequences of regions. I will illustrate phase diagram construction on the piecewise linear equations (2.2) that PLR chooses to approximate the Lienard equation (2.3), as shown in the previous chapter.

3.1 Local Analysis of Linear Equations

PLR employs the standard theory of linear equations [19, ch. 1–6] to derive the phase diagram of a second-order linear system. First, it calculates the two eigenvalues of the system. Their disposition in the complex plane determines the structure of the phase diagram. A zero eigenvalue indicates a degenerate system, which PLR cannot handle. Figure 3.1 lists the possibilities for a nondegenerate system and shows the associated phase diagrams. The unique fixed point of the system is placed at the origin of each phase diagram. The first row of diagrams shows the three types of qualitative behavior for distinct real eigenvalues: a *sink*

Disposition of Eigenvalues

real					complex		
unequal			equal		real part		
both neg	one each	both pos	neg	pos	neg	zero	pos
real sink	saddle	real source	improper sink	improper source	spiral sink	center	spiral source

Phase Diagrams

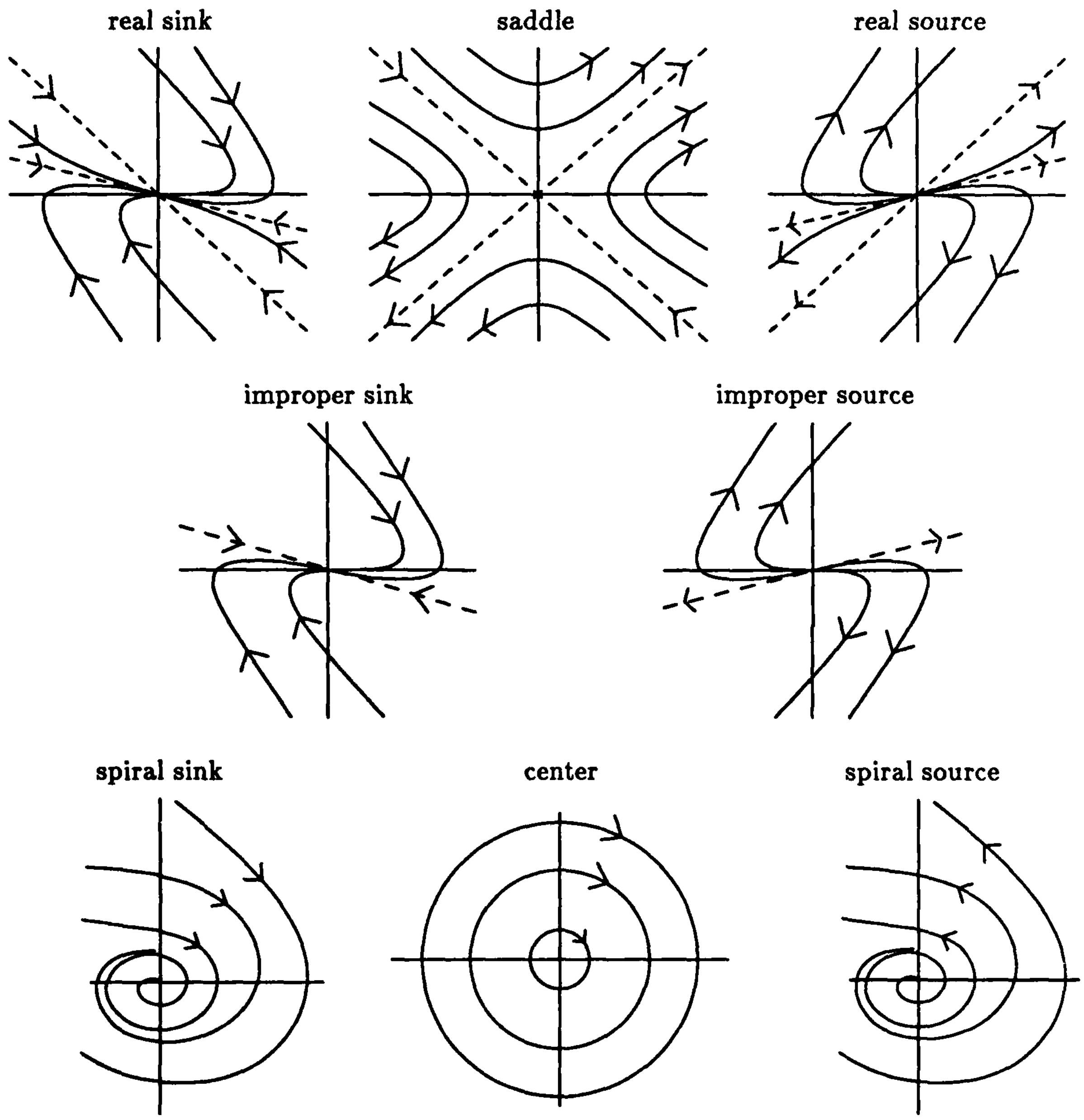


Figure 3.1: Prototypical phase diagrams of second-order linear systems; actual diagrams may be distorted and/or transposed by a linear change of coordinates.

when both are negative, a *saddle* when they have opposite signs, and a *source* when both are positive. Two asymptotes, indicated by dashed lines, partition the trajectories into disjoint sets. The second row shows *improper* sinks and sources, which arise when the eigenvalues are real and identical. They resemble sinks and sources, but have a single asymptote. The third row shows the complex cases. The two eigenvalues are complex conjugates, so they have the same real part. A *spiral* sink or source occurs when the real part is negative or positive and a *center* when it is zero.

3.2 Region Formation

PLR partitions the phase space for a system of two piecewise linear equations into maximal regions on which both equations are linear. The boundaries between regions consist of the lines $x = x_i$ and $y = y_i$ at which the piecewise linearizations in the x and y coordinates change. PLR calculates the fixed points and eigenvalues of the linear systems associated with the regions. It subdivides regions that have real eigenvalues along their asymptotes. It determines the local phase diagrams for the regions from their eigenvalues and fixed points, as explained in the previous section. For example, Figure 3.2 shows the regions, labeled A through E, that PLR constructs for the piecewise linear Lienard equations (2.2). The thick line denotes the boundary $x = -1/2$ at which the piecewise linearization changes. The left-hand linear system has a saddle (labeled s) at the point $(-1, 0)$ and the right-hand system has a spiral sink (labeled ss) at the origin. PLR splits the left-hand region into subregions A through D along the asymptotes of the saddle, denoted by dashed lines.

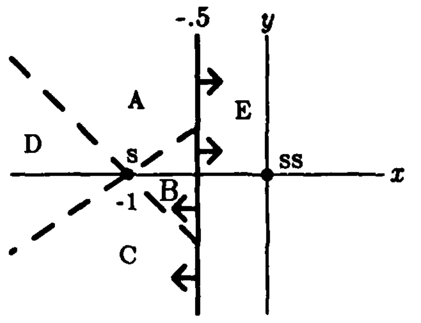


Figure 3.2: Phase diagram for the Lienard example

3.3 Transition Analysis

PLR determines whether local trajectories of a region access an adjacent region by examining their behavior on the boundary between the two. It only examines boundaries between linear regions of the piecewise linear system. Trajectories cannot cross boundaries within linear regions (produced by asymptotes) because linear systems have unique solutions for each initial value. They can only access adjacent regions because linear systems have continuous solutions. PLR handles boundaries of the form $y = u(x)$ and $x = u(y)$ with u a closed-form differentiable function and the free variable ranging over an interval. Multi-valued mappings must be split into their branches.

For a trajectory to cross from region R to S via boundary u, its tangent t at the intersection point with u must form an acute angle with the normal n, as shown in Figure 3.3. In algebraic terms, the inner product $t \cdot n$ must be positive. Conversely, the entire trajectory remains in R if $t \cdot n \leq 0$ on u. PLR calculates $t = a \begin{bmatrix} x \\ y \end{bmatrix}$ from the linear system on R

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = a \begin{bmatrix} x \\ y \end{bmatrix} + b \tag{3.1}$$

and derives n from Figure 3.4. It tries to prove $t \cdot n \leq 0$ on u using the BOUNDER inequality reasoner described in Chapter 7. It assumes that R accesses S if the proof fails. This assumption sometimes introduces spurious transitions because no algorithm can prove all true inequalities. When the proof succeeds, PLR deduces that R does not access S.

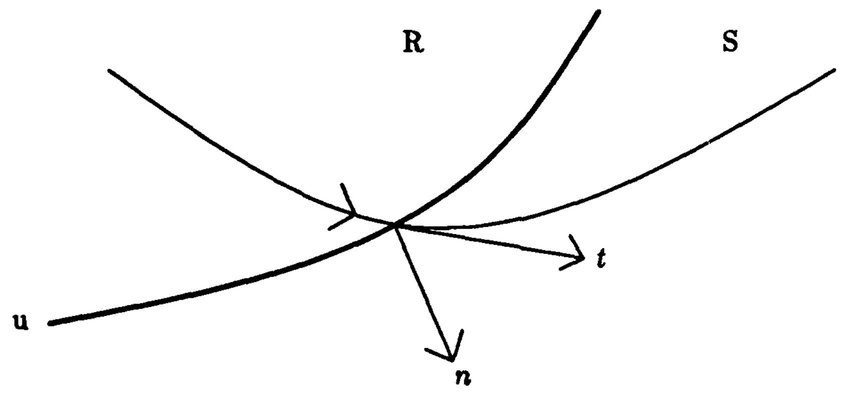


Figure 3.3: Trajectory crossing from R to S via u: the tangent t at the crossing point must form an acute angle with n, the normal to u that points into S.

$y = u(x)$ boundary		$x = u(y)$ boundary	
$y > u(x)$ on R	$y < u(x)$ on R	$x > u(y)$ on R	$x < u(y)$ on R
$\begin{bmatrix} u'(x) \\ -1 \end{bmatrix}$	$\begin{bmatrix} -u'(x) \\ 1 \end{bmatrix}$	$\begin{bmatrix} -1 \\ u'(y) \end{bmatrix}$	$\begin{bmatrix} 1 \\ -u'(y) \end{bmatrix}$

Figure 3.4: Formulae for the normal n to u pointing from R to S

In the piecewise linear systems that PLR constructs, all the boundaries between regions are lines. This makes transition analysis much easier and precludes spurious transitions. The inequality $t \cdot n \leq 0$ is linear in x and y, hence resolvable by BOUNDER, because n is constant and t contains only linear terms. The line $t \cdot n = 0$ divides the plane into a region on which the inequality holds and a region on which it does not. When u intersects this line, it splits into one line segment that trajectories cross and another that they do not (Figure 3.5a). When u lies entirely in one region, either all or no trajectories cross (Figure 3.5b). In the Lienard example (Figure 3.2), $t \cdot n \leq 0$ evaluates to $y \leq 0$ on the boundary between A and E and to $y \geq 0$ between E and B. The line $y = 0$ delimits the segments of the boundaries that trajectories cross. Arrows mark the middles of these segments.

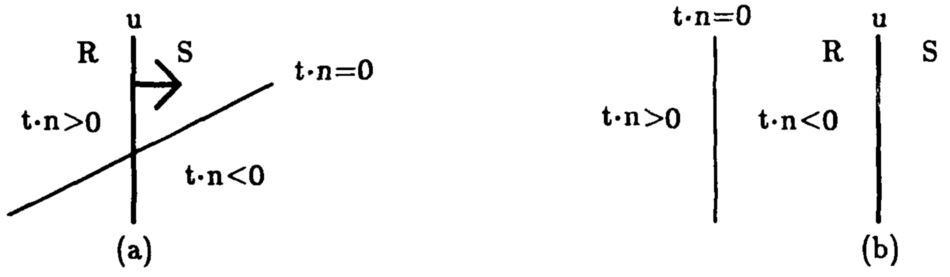


Figure 3.5: (a) $t \cdot n = 0$ intersects u (b) it does not

3.4 Transition Graphs

PLR records the results of transition analysis in a transition graph. Nodes represent regions and fixed points. Links between regions represent accessibility relations. Self-loops and links from regions to fixed points represent unbounded and bounded trajectories that remain in one region forever. Unbounded trajectories can remain in a region forever if they move monotonically in directions where it is unbounded. Bounded trajectories must asymptotically approach a sink or periodically rotate around a center. Figure 3.6 summarizes the semantics of transition graphs. Figure 3.7 shows the transition graph for the Lienard example.

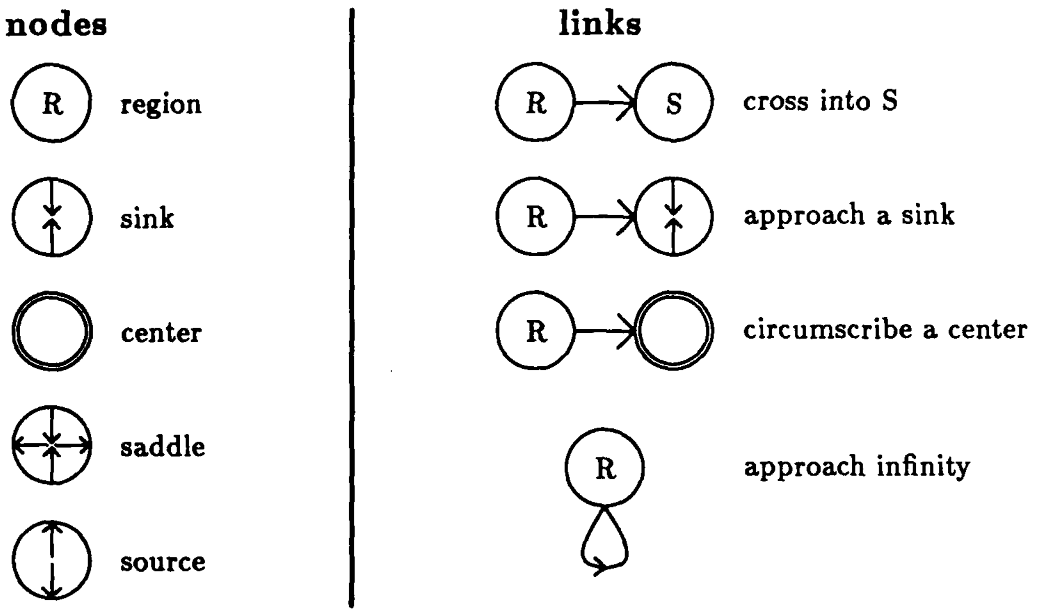


Figure 3.6: *Semantics of transition graphs*

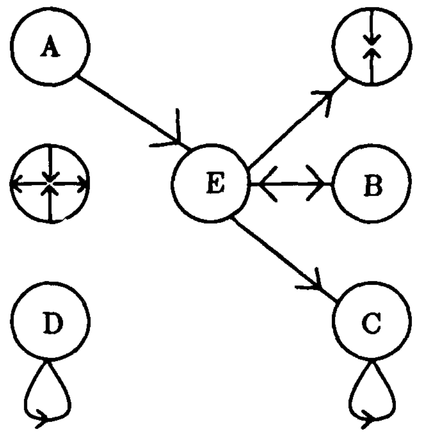


Figure 3.7: Transition graph for the Lienard example

PLR does not construct nodes for *pseudo fixed points:* fixed points that lie outside the regions in which their linear systems are in effect. Trajectories cannot remain in a region whose linear system has a pseudo sink forever; they must leave it to approach the fixed point. Section 4.1 describes a system with pseudo sinks.

3.5 Case Analysis

The transition graph for a system contains every path that trajectories traverse, but may also contain *vacuous paths* that no trajectories traverse. A path R-S-T is vacuous when none of the trajectories that enter region S from R crosses into T. For example, the path B-E-B in Figure 3.7 is vacuous because trajectories that enter E from B remain there, spiraling toward the origin, and cannot return to E. Some trajectories do cross from E to B, but not those that originate in B. In other words, the link E-B is conditioned by the predecessor of E. Similarly, path B-E-C is vacuous. Case analysis converts conditional links to sets of simple links. It eliminates vacuous paths by deleting simple links that prove untraversable.

PLR represents the dependence of transitions on predecessors by partitioning regions of out-degree greater than one into subregions of out-degree one. Each subregion consists of all points through which trajectories cross into a specific successor. Subregions initially inherit the predecessors of their parents. Let region S above have subregions $S_1, \ldots, S_k$. The conditional link S-T in the original transition graph translates into one of the simple links S_1-T,. . . ,S_k-T. The refined graph for the piecewise Lienard equations appears in Figure 3.8a. The conditional links E-B and E-C translate into the simple links E2-B and E3-C.

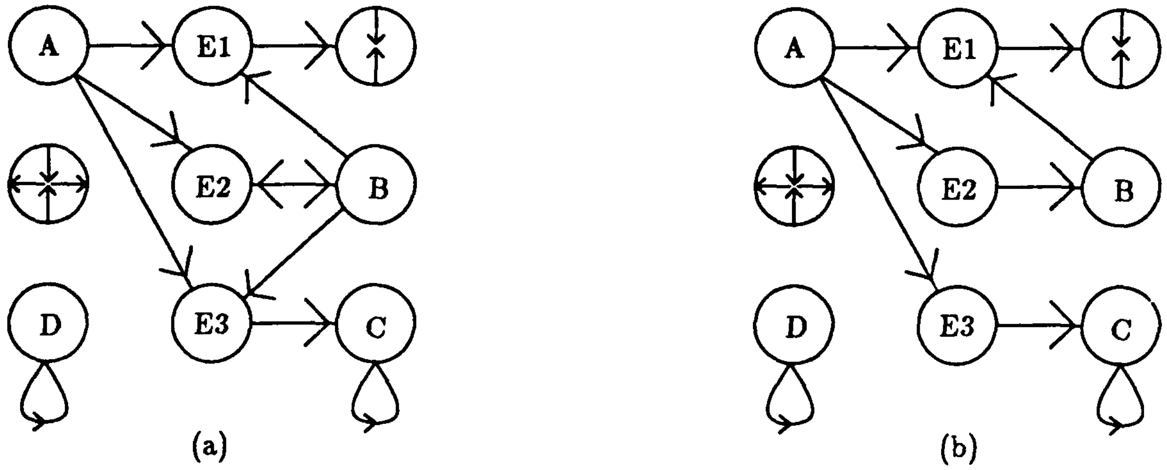


Figure 3.8: (a) Region splitting and (b) case analysis for the piecewise Lienard equations

Refining a transition graph guarantees that every vacuous path contains an untraversable link by the definition of subregions. A link R-S_i is untraversable if S_i and the boundary between R and S are disjoint. PLR could detect all untraversable links if it could precisely

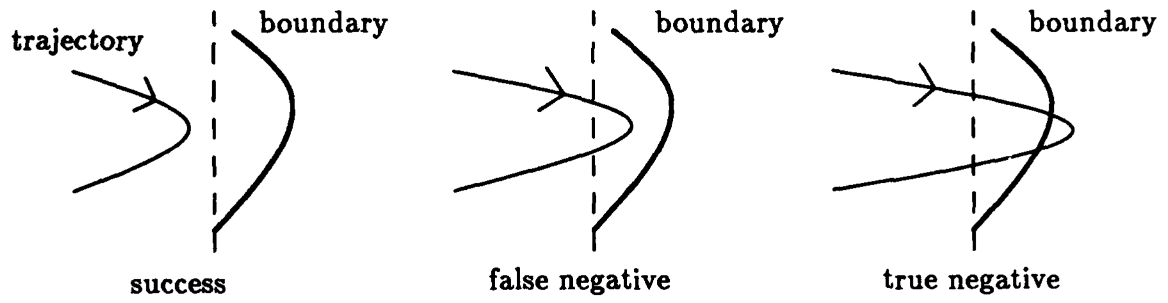


Figure 3.9: Rule-out example: points to the left of the dashed line are disjoint from the boundary in the x dimension.

determine the boundaries of subregions. It cannot do so in practice because the boundaries are the solutions to complicated, generally unsolvable, algebraic and transcendental equations. Instead, it embeds each subregion in a larger set consisting of all points that satisfy certain algebraic constraints. If all points on the boundary between R and S satisfy the constraint for S_i, the boundary and S_i must be disjoint. PLR deletes every link for which it succeeds in proving this. In the Lienard example, it deletes the links from B to E2 and E3, thus eliminating all vacuous paths (Figure 3.8b).

When refining a transition graph, PLR does not recursively split regions that have out-degree greater than one because it could not characterize the boundaries of the results. Creating recursive subregions would increase the complexity of transition graphs without improving their predictive power. In the Lienard example, A has out-links to E1, E2, and E3, but [illegible] annot describe the subsets of A that access the individual Ei.

The constraints for a subregion rule out points whose trajectories are disjoint from the boundary with the subregions' successor in either coordinate. I chose this constraint over the weaker constraint that trajectories are disjoint from the boundary because it reduces directly to an inequality that PLR can evaluate. The two constraints are equivalent when every boundary between linear regions consists of a line parallel to an axis, as is the case in the approximations that PLR constructs. Subtler constraints may be required in future implementations that construct nonlinear boundaries. Figure 3.9 shows three applications of PLR's constraint: one where it succeeds, one where it fails although the weaker condition holds, and one where the weaker condition does not hold.

Let boundary b separate a subregion S_i from its successor. Let p be a point whose x

component is less than the lower bound l of the projection of b on the x axis. (These are the points to the left of the dashed lines in Figure 3.9.) The constraint for S_i rules out p if the x component of the trajectory through p

R1a) decreases monotonically, or

R2a) has an upper bound less than l, or

R3a) decreases toward a local minimum outside S.

Figure 3.10 illustrates these cases. The same rules apply to the y coordinate. Analogous rules R1b–R3b apply to a point one of whose components is greater than the upper bound of b.

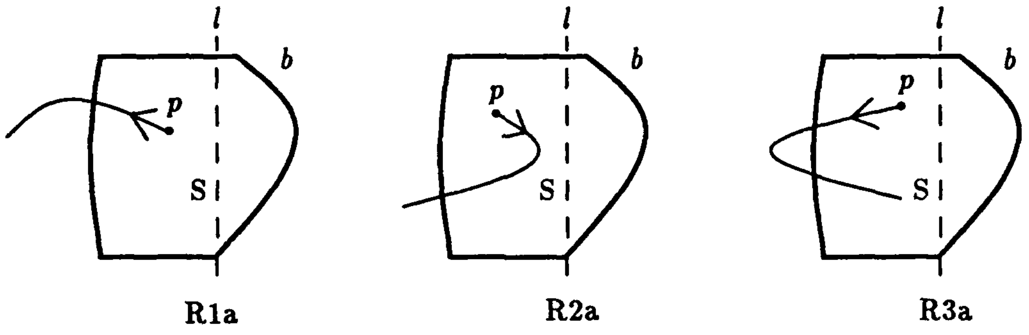


Figure 3.10: Rule-out constraints for a point p to the left of a boundary

The exact form of the constraint for a subregion depends on the linear system of the containing region. In a system with real eigenvalues α_1 and α_2, trajectories have components

$$c_1 e^{\alpha_1 t} + c_2 e^{\alpha_2 t} + k \tag{3.2}$$

with c_1 and c_2 linear functions of the initial value. The c_i have fixed signs on each region because the lines $c_i = 0$ where they change sign are asymptotes, which lie entirely on boundaries between regions. Equation (3.2) is either monotonic or unimodal depending on the signs of the c's and α's, as shown in Figure 3.11. For initial values less than boundaries, PLR constructs an identically false constraint in case 1 of the figure and an identically true constraint in case 2. In case 3, it instantiates R3a and in case 4, it instantiates R1a and R2a. It handles initial values greater than boundaries analogously.

	case	behavior
1.	$c_1 > 0, \alpha_1 > 0, c_2 > 0, \alpha_2 > 0$ or $c_1 > 0, \alpha_1 > 0, c_2 < 0, \alpha_2 < 0$ or $c_1 < 0, \alpha_1 < 0, c_2 < 0, \alpha_2 < 0$	monotone increasing
2.	$c_1 < 0, \alpha_1 > 0, c_2 < 0, \alpha_2 > 0$ or $c_1 < 0, \alpha_1 > 0, c_2 > 0, \alpha_2 < 0$ or $c_1 > 0, \alpha_1 < 0, c_2 > 0, \alpha_2 < 0$	monotone decreasing
3.	$c_1 > 0, \alpha_1 > 0, c_2 < 0, \alpha_2 > 0$ or $c_1 > 0, \alpha_1 > 0, c_2 > 0, \alpha_2 < 0$ or $c_1 < 0, \alpha_1 < 0, c_2 > 0, \alpha_2 < 0$	unimodal; minimum at $\frac{\log(-c_2\alpha_2/c_1\alpha_1)}{\alpha_1 - \alpha_2}$
4.	otherwise	unimodal; maximum at $\frac{\log(-c_2\alpha_2/c_1\alpha_1)}{\alpha_1 - \alpha_2}$

Figure 3.11: Behavior of $c_1e^{\alpha_1 t} + c_2e^{\alpha_2 t} + k$ for real α_1, α_2 with $\alpha_1 > \alpha_2$

Systems with complex eigenvalues have components

$$ce^{\alpha t}\sin(\beta t + \gamma) + k \tag{3.3}$$

with $c > 0$ and γ functions of the initial point. PLR does not instantiate rule R1 (a or b) because these functions are never monotonic. It instantiates R3 with the first extremum of equation (3.3). When $\alpha \leq 0$, the extrema decrease monotonically in absolute value, so the global maximum and minimum occur at the first two. PLR uses these values in R2. When $\alpha > 0$, it invokes BOUNDER (Ch. 7) on equation (3.3) to calculate bounds for R2.

PLR obtains additional rule-out constraints for the linear system

$$\begin{cases} x' = ax + by \\ y' = cx + dy \end{cases} \tag{3.4}$$

from the "energy" function

$$V_e(x, y) = (ad - bc)x^2 + (ax + by)^2 \tag{3.5}$$

described in Section 5.2. Equation (3.4) describes a system whose fixed point occurs at the origin. PLR reduces a general system with fixed point (x_0, y_0) to this case by substituting $x - x_0$ for x and $y - y_0$ for y. The function V_e increases, is constant, or decreases along all trajectories depending on whether

$$\dot{V}_e = 2(d + a)(ax + by)^2 \tag{3.6}$$

is positive, zero, or negative. $\dot{V}_e$ is positive for sources, zero for centers, negative for sinks and nonzero for saddles. In the nonpositive cases, the trajectory through p cannot cross boundary b if $V_e(p) > V_e(q)$ for every q in b. In the nonnegative cases, it cannot cross if $V_e(p) < V_e(q)$. In the Lienard example, PLR rules out the links from B to E2 and E3 with an energy constraint.

Transitions from regions to sinks and centers, representing trajectories that remain near fixed points forever, have different rule-out constraints than transitions between regions. A trajectory cannot stay in a region forever if either component has an extremum outside the region or increases monotonically in a direction where the region is bounded. This constraint enables PLR to delete the links from A, C, F, and G to D1 in the transition graph for the undamped pendulum (Figure 1.4).

3.6 Correctness and Computational Complexity

A *sound* transition graph for a piecewise linear system contains a link A $\rightarrow$ B if A accesses B, that is some trajectory crosses from region A to region B. A *complete* transition graph contains A $\rightarrow$ B only if A accesses B. Piecewise analysis derives sound transition graphs for arbitrary piecewise linear systems. It constructs a sound initial graph for a system by linking each region with all its neighbors. Trajectories cannot cross between disjoint regions because of their continuity. Transition analysis and case analysis preserve soundness by only deleting links that they prove uncrossable. These conclusions rest upon the soundness of BOUNDER, which I prove in Chapter 7. Piecewise analysis is incomplete. It retains uncrossable links when BOUNDER fails on true inequalities.

Piecewise analysis is sound and complete for systems in which the boundaries between regions are lines, rather than general closed-form analytic curves. The inequalities that it must resolve are linear in the state variables. They are resolvable by BOUNDER unless externally-specified nonlinear constraints exists. Hence, PLR is sound and complete, since the regions that it constructs have linear boundaries. Completeness of PLR with respect to other classes of nonlinear systems is an open problem.

In general, the soundness and completeness of piecewise analysis hinges on its ability to resolve inequalities. One could implement an unsound and complete version by choosing an unsound and complete inequality prover and modifying transition analysis and case analysis to delete every link that they cannot prove crossable. However, no implementation can be sound and complete because no algorithm can prove all inequalities between extended elementary functions, as shown by Richardson [36].

The computational complexity of PLR depends on that of BOUNDER, which is exponential in the length of its input. If all externally-specified constraints are linear, this can be reduced to polynomial time by replacing BOUNDER with the Karmarkar algorithm. I have not implemented that complicated algorithm.

3.7 Other Phase Spaces

The preceding sections of this chapter presuppose a planar phase space in which regions are adjacent if and only if they share a common boundary. Some systems have clearer, simpler descriptions in other phase spaces. Periodic systems call for phase spaces that equate equivalent states. For example, the natural space for the pendulum equations

$$\begin{cases} \theta' = \omega \\ \omega' = -\frac{g}{l} \sin\theta \end{cases} \tag{3.7}$$

consists of a cylinder (Figure 3.12a), each of whose points (θ, ω) represents the infinite set of equivalent states

$$\{(\theta + 2\pi n, \omega) : n = \ldots, -2, -1, 0, 1, 2, \ldots\}. \tag{3.8}$$

By the same token, a torus is the natural space for a system of two periodic equations.

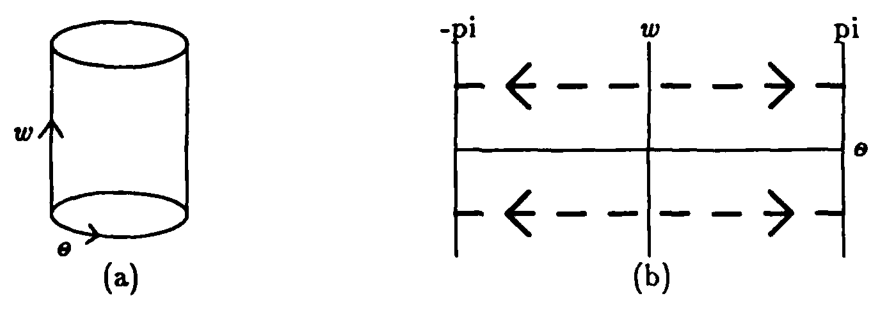


Figure 3.12: (a) a cylindrical phase space and (b) the corresponding adjacency map

PLR represents cylinders, tori, and other non-planar phase spaces by generalizing the definition of adjacency. It permits any pair of regions to be adjacent along any pair of boundaries, instead of requiring them to *share a common* boundary. *According* to this scheme, any two regions can be adjacent, but no region can be adjacent to two regions along a single boundary. Each adjacency instance includes a homeomorphism (a continuous function that has a continuous inverse) between the corresponding pair of connected boundaries, as Figure 3.13 illustrates. The identity function on a boundary, used implicitly so far, defines planar adjacency. The function

$$(\theta,\omega) \rightarrow (\theta - 2\pi, \omega) \tag{3.9}$$

from the line $\theta = \pi$ to $\theta = -\pi$ defines the cylindrical phase space for the pendulum (Figure 3.12b), whereas the function

$$(\theta,\omega) \rightarrow (\theta - 2\pi, \omega - 2\pi) \tag{3.10}$$

defines a torus. PLR *can construct cylinders and tori automatically.* Other types of adjacency must be specified by the user.

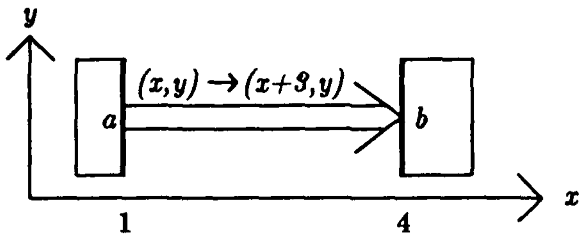


Figure 3.13: A generalized adjacency instance between a and b

Generalized adjacencies do not affect linear analysis or transition analysis because both occur within individual regions. Case analysis applies rule-out constraints to the images of boundaries instead of the boundaries themselves. In the remainder of the thesis, all adjacencies are identity maps on shared boundaries, unless stated otherwise.

Chapter 4

Examples

The first three sections of this chapter demonstrate piecewise analysis on several differential equations that model important physical laws and engineering devices. PLR constructs the piecewise linear approximations from the given equations. In the final section, I discuss a proposal by Kalman for engineers to analyze piecewise linear models of servo-mechanisms. I show that PLR handles all the examples in his paper and draws the desired conclusions.

4.1 Van der Pol Equations

Van der Pol equations often arise in oscillatory dynamic systems. Figure 4.1 depicts a simple example from network theory: a capacitor, an inductor, and a nonlinear resistor connected in series. By Kirchoff's laws, the current through the circuit obeys the equation

$$i'' + \frac{k}{L}(i^2 - 1)i' + \frac{1}{LC}i = 0, \tag{4.1}$$

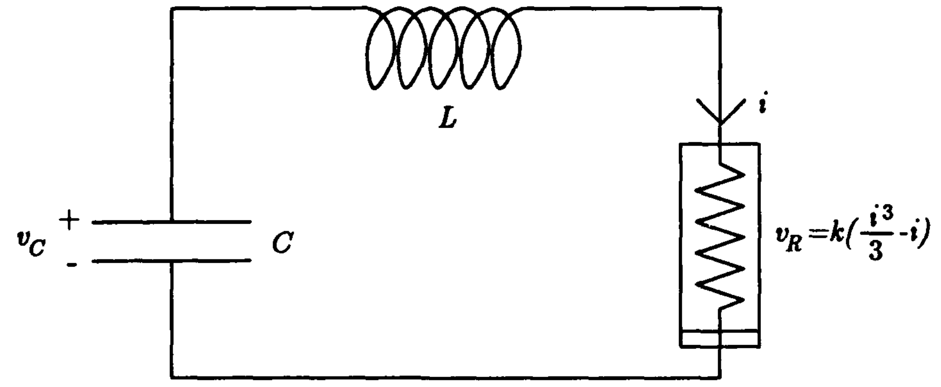


Figure 4.1: A circuit governed by van der Pol's equation

with C the capacitance, L the inductance, and k a scaling factor. The symbolic parameters here and throughout this chapter are declared positive. Intuitively, the system oscillates because the nonlinear resistor adds energy to the circuit at low currents and drains energy at high currents. PLR reformulates the nonlinear term $(i^2-1)i'$ as $(i^3/3-i)'$ and approximates $i^3/3-i$ with the piecewise linear function

$$f(i) = \begin{cases} 2(i+\sqrt{3})/3(\sqrt{3}-1) & \text{for} \quad i < -1 \\ -2i/3 & \text{for} \quad -1 \le i \le 1 \\ 2(i-\sqrt{3})/3(\sqrt{3}-1) & \text{for} \quad i > 1 \end{cases} \tag{4.2}$$

by the method shown in chapter 2. Figure 4.2 illustrates the original term and its piecewise linear approximation. The analysis of the resulting system

$$\begin{cases} i' = y \\ y' = -\frac{1}{LC}i - \frac{k}{L}f'(i)y \end{cases} \tag{4.3}$$

follows the general pattern described in the previous chapter, although the symbolic parameters k, L, and C complicate the process somewhat. PLR must consider two cases, corresponding to whether the characteristic equation has real or complex roots. The fact that $f'(i)$ is undefined on the boundaries between linear regions does not pose a problem; it suffices for the analysis that the one-sided derivatives exist.

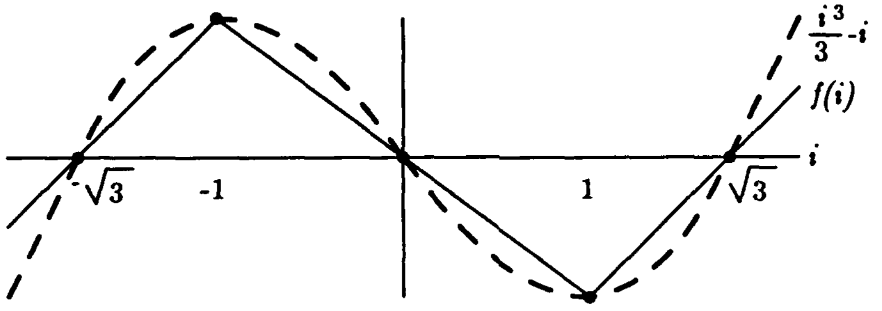


Figure 4.2: Piecewise Linearization of $i^3/3-i$

The characteristic equation has complex roots for small values of k and real roots for large values.[1] In physical terms, these cases represent under-damped and over-damped systems. Figures 4.3 and 4.4 show the phase diagrams and transition graphs for small and large k. In

[1]Throughout the chapter, I ignore the case of a zero discriminant, which produces an improper node, because of its rarity and similarity to the nonzero cases. PLR handles it analogously.

both cases, the regions between the boundaries $i = \pm 1$ have a source at $(0,0)$ (labeled sr and rr for spiral and real repeller) and the regions outside have a pseudo sink at $(0,0)$. Transition analysis determines that trajectories cross the boundaries from left to right at points above the origin and from right to left below. For small k, case analysis rules out transitions from B to A2 by rule R3a and from C to A1 by rule R3b. The graph consists of a single cycle around the origin: C-A2-B-A1. For large k, case analysis rules out transitions from A and B to E1 by R1a and from G and J to C1 by R1b (p. 27). The graph contains a single cycle in which all paths terminate: C2-D-E2-H. Both graphs show that all trajectories spiral around the origin, but tell nothing of whether they move inward, move outward, or wobble around. Global analysis, described in Chapter 5, proves that they converge to a unique limit cycle.

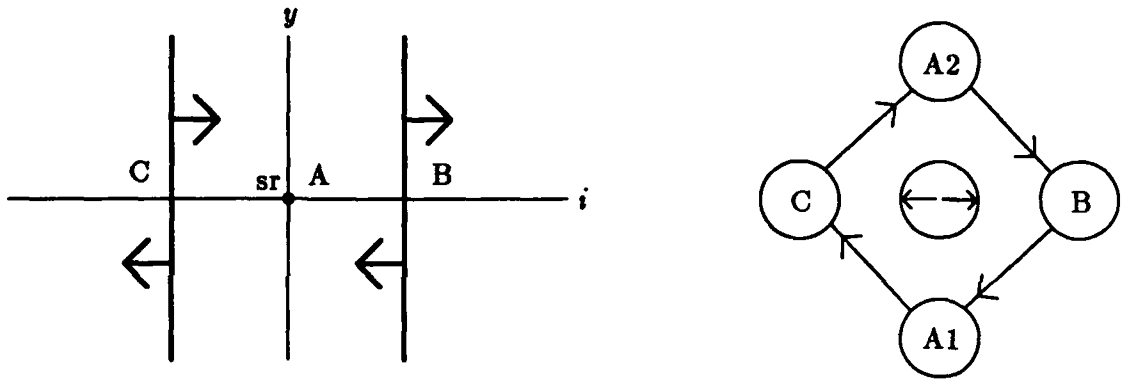


Figure 4.3: Phase diagram and transition graph for the van der Pol equation: small k

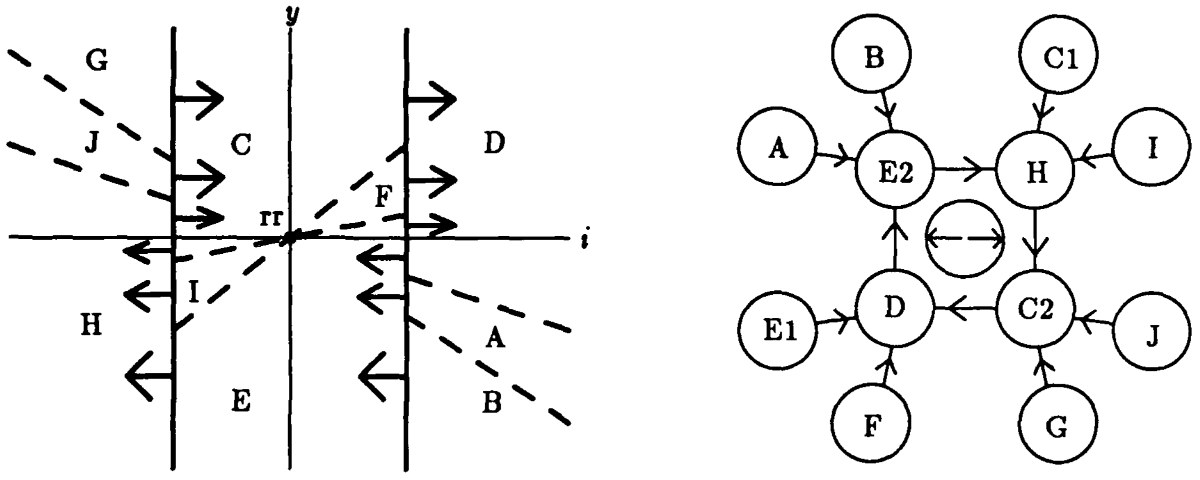


Figure 4.4: Phase diagram and transition graph for the van der Pol equation: large k

4.2 Gravitational Pendulum

The standard model for the gravitational pendulum is

$$\theta'' + \frac{\mu}{m}\theta' + \frac{g}{l}\sin\theta = 0, \tag{4.4}$$

with θ the angle between the arm and the vertical, l the length of the (weightless rigid) arm, m the mass of the bob, g the gravitational constant, and μ the damping coefficient (Figure 4.5). In the introduction, I discussed informally the special case of an undamped pendulum, obtained by setting $\mu = 0$. The analysis in this and the following sections takes place in the cylindrical phase space obtained by identifying the lines $\theta = -\pi$ and $\theta = \pi$. The symmetry and 2π periodicity of the pendulum equation make this space natural.

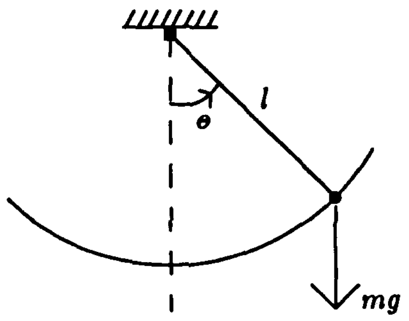


Figure 4.5: A gravitational pendulum

PLR obtains the approximate pendulum equations

$$\begin{cases} \theta' = \omega \\ \omega' = \dfrac{2g}{\pi l} f(\theta) - \dfrac{\mu}{m}\omega \end{cases} \tag{4.5}$$

with

$$f(\theta) = \begin{cases} \theta + \pi & \text{for } -\pi \le \theta < -\frac{\pi}{2} \\ -\theta & \text{for } -\frac{\pi}{2} \le \theta \le \frac{\pi}{2} \\ \theta - \pi & \text{for } \frac{\pi}{2} < \theta \le \pi \end{cases} \tag{4.6}$$

by piecewise linearizing $\sin\theta$ (Figure 1.2). The first and third systems have saddles at $(\pm\pi, 0)$ because their characteristic roots

$$-\frac{\mu}{2m} \pm \sqrt{\frac{\mu^2}{4m^2} + \frac{2\pi g}{l}} \tag{4.7}$$

are real and of opposite signs. The two saddles coincide in the cylindrical phase space. The second system has a center at the origin in the undamped case. It has a real or spiral sink in the damped case depending on whether it is over- or under-damped, that is whether the discriminant

$$\frac{\mu^2}{4m^2} - \frac{2\pi g}{l} \tag{4.8}$$

is positive or negative.

The phase diagrams and transition graphs for the three cases appear in Figures 4.6, 4.7, and 4.8. The undamped pendulum can oscillate below the horizontal (D1-(0, 0)), oscillate above the horizontal (C-D2-G-D4), or spin around forever (F-D5-B and A-D3-E). Case analysis rules out some links, such as C-D3 and G-D5, by energy constraints and others, such as C-D4 and G-D2, by rules R3a and R3b. The under-damped pendulum transition graph contains analogues of the undamped pendulum's paths along with links from higher energy paths to lower energy ones, which no longer violate energy constraints. The pendulum can change from spinning to oscillating and from oscillating above the horizontal to oscillating below. The over-damped pendulum can spin or approach the origin, but not oscillate. The transition graphs do not establish that every damped pendulum trajectory approaches the origin asymptotically. The next chapter shows how global analysis proves this.

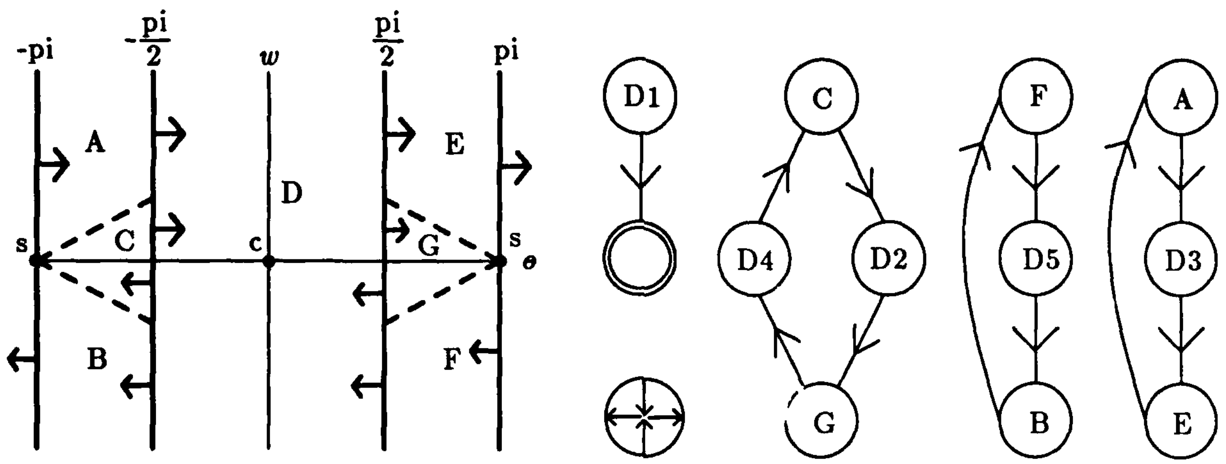


Figure 4.6: Phase diagrams and transition graph for the undamped pendulum

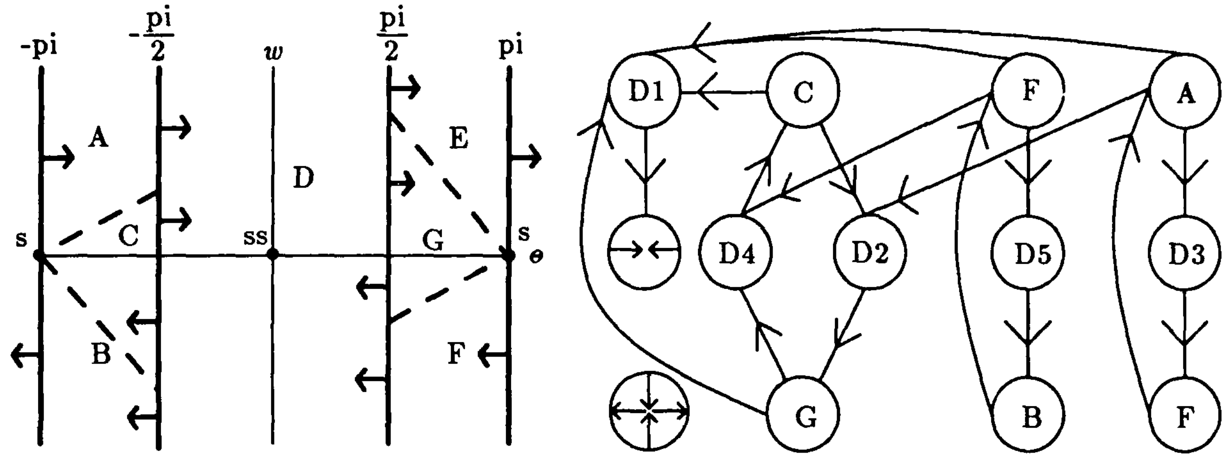


Figure 4.7: Phase diagrams and transition graph for the under-damped pendulum

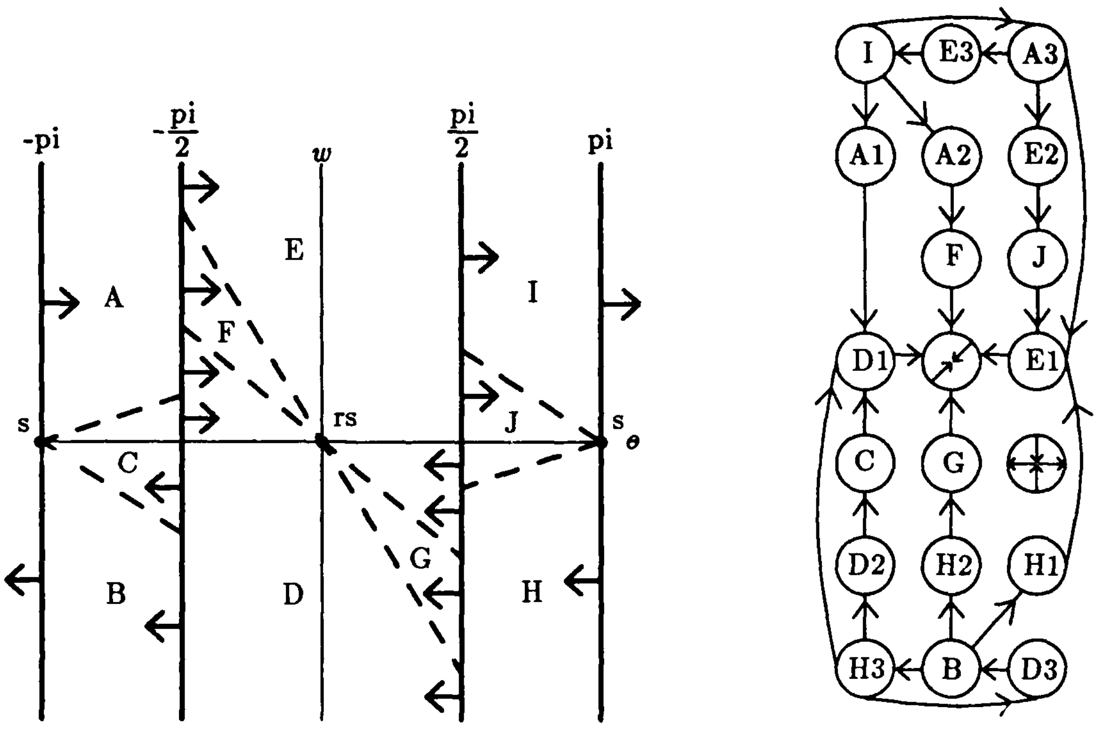


Figure 4.8: Phase diagrams and transition graph for the over-damped pendulum

4.3 Magnetic Pendulums

The simple pendulum model presupposes that the force on the bob is independent of the bob's location. The model for a variable force, such as attraction between a magnet and a metallic pendulum (Figure 4.9), is more complicated. The magnetic force points from the bob to the magnet with magnitude inversely proportional to the distance between them. I will ignore gravitation (even though PLR can handle it), since it complicates the model without producing new behavior. The resulting equation is

$$\theta'' + \frac{\mu}{m}\theta' + wl(r+1)f(\theta) = 0, \tag{4.9}$$

with

$$f(\theta) = (r^2 + 2r + 2 - 2r\cos\theta)^{-3/2}\sin\theta, \tag{4.10}$$

r a scaling factor, w the coefficient of magnetic attraction, and the other constants as in equation (4.4). Straightforward analysis establishes that f is odd and bimodal with a positive maximum in the interval $(0, \pi/2)$ and a negative minimum in $(-\pi/2, 0)$. These facts enable PLR to piecewise linearize f, as shown in Figure 4.10. The analysis of the corresponding equations is unaffected by the exact, r-dependent locations and values of the extrema. It is identical to that of the piecewise linear gravitational pendulum equations (4.5).

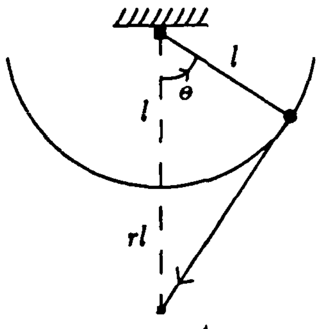


Figure 4.9: A metallic pendulum attracted by a magnet

The magnetic and gravitational pendulums have the same qualitative behavior, even though the restoring force varies in the former and remains constant in the latter. Significant differences manifest themselves in the presence of competing forces. Two constant forces, such as gravitation, behave the same as one: their vector sum. Yet replacing the single

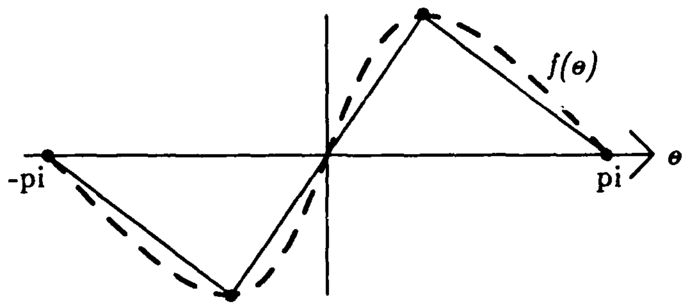


Figure 4.10: *Piecewise linearization of $f(\theta)$*

magnet in Figure 4.9 with a horseshoe magnet, as shown in Figure 4.11, completely changes the pendulum's motion. The horseshoe pendulum obeys the equation

$$\theta'' + \frac{\mu}{m}\theta' + wl(r+1)g(\theta) = 0 \tag{4.11}$$

with $g(\theta) = f(\theta + a) + f(\theta - a)$ and a the angle between each pole and the vertical. The function g has the same general shape as f when r is large with respect to a and μ (Figure 4.12a). A distant horseshoe magnet causes the same motion as a single magnet because the bob cannot distinguish between its poles. As r shrinks, g's slope at the origin grows until it becomes positive and two additional extrema appear (Figure 4.12b). The bob begins to feel the effect of each pole separately at this point. I had to perform this analysis and construct the piecewise linearizations by hand because QMR cannot handle g, as explained in Chapter 2.

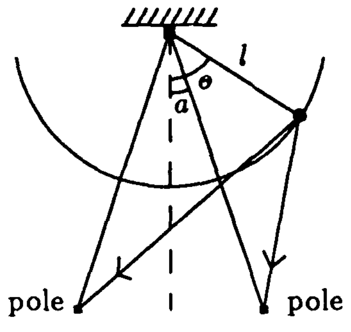


Figure 4.11: A metallic pendulum attracted by a horseshoe magnet

Figure 4.13 contains the phase diagram and transition graph of an undamped pendulum for a nearby horseshoe magnet. The pendulum can spin (A-D3-E-I3-J and K-I5-F-D5-B), oscillate between both poles (D3-E-I5-F and C-D3-E-I2-L-I5-F-D4), oscillate around the left

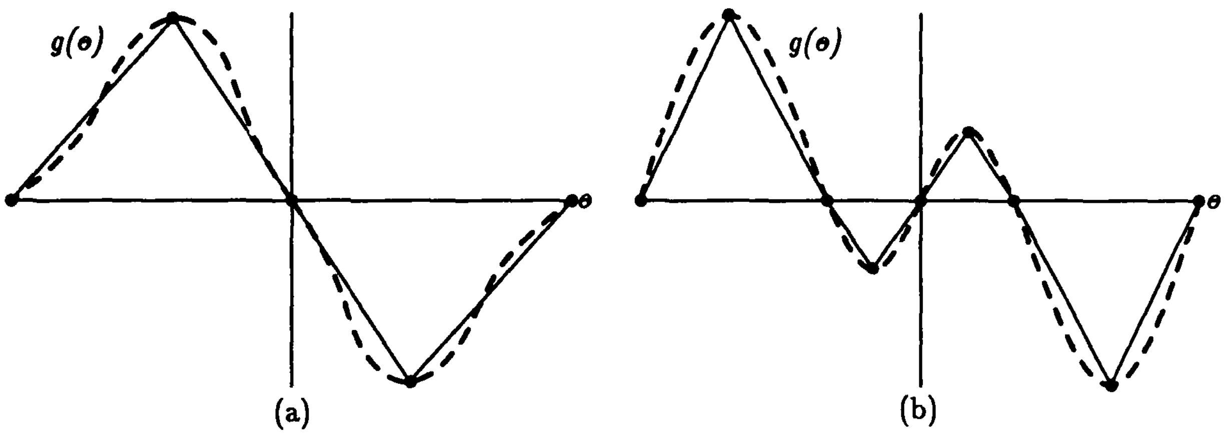


Figure 4.12: Piecewise linearizations of $g(\theta)$ for fixed a, μ: (a) large r (b) small r

pole (D2-G and D1-(0,0)), or oscillate around the right pole (I4-H and I1-(0,0)). Saddles appear at $(0, \pm\pi)$ and $(0,0)$ where the poles cancel each other. A center appears where each pole is strongest. Additional spurious paths, such as C-D3-E-I3-J, are ruled out by the methods described in Section 5.2. Adding damping to the model changes the centers to sinks. The transition graph contains the previous links along with links from higher energy paths to lower energy ones. Adding gravitation changes the origin to a sink. The two existing sinks persist when the poles of the magnet can overcome the weight of the bob, but disappear when they cannot. Global analysis discovers that the pendulum's oscillations damp out in the presence of friction and continue forever in its absence.

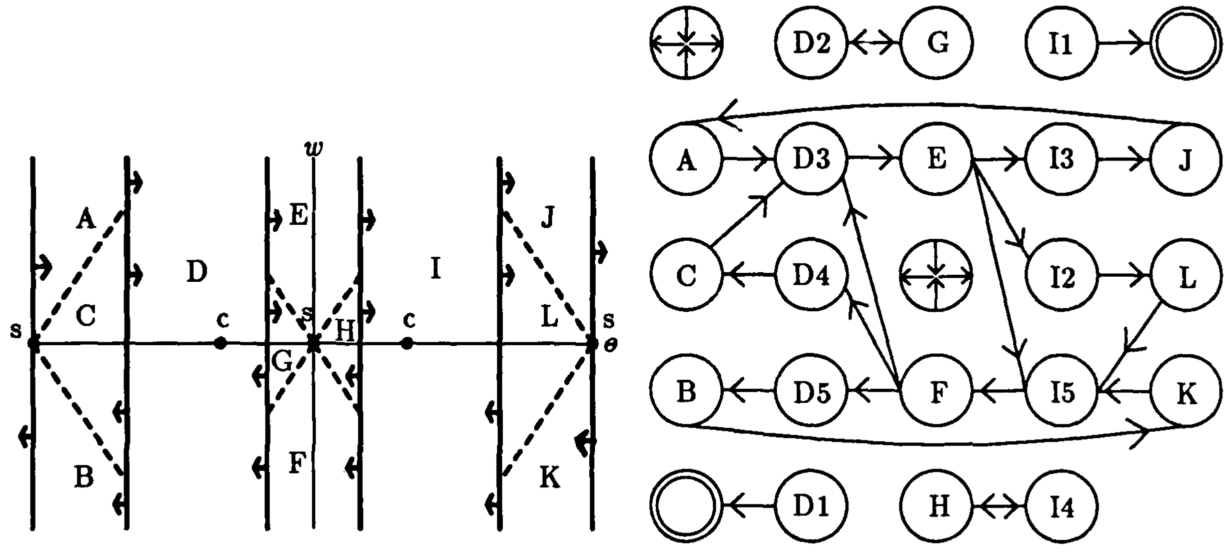


Figure 4.13: Transition graph and regions for the horseshoe magnet pendulum

4.4 Automatic Control Systems

Kalman [23] derives the qualitative behavior of feedback control systems by constructing phase diagrams of piecewise linear approximations to the differential equations that govern them. Like PLR, he builds a composite phase diagram for a piecewise system by combining the local diagrams of its linear regions. He combines the local diagrams by inspection; the concepts of transition analysis and case analysis are not developed. Kalman demonstrates the usefulness of his approach for engineers with several examples involving servo-mechanisms. PLR handles all the examples in his paper and draws the desired conclusions.

In his first example, Kalman models a positional servo-mechanism by the equation

$$e'' + \frac{1}{T}e' + \frac{1}{T}f(e) = 0 \tag{4.12}$$

containing a piecewise linear gain element

$$f(e) = \begin{cases} k_1 e & \text{for} \quad |e| < e_0 \\ k_2 e & \text{for} \quad |e| > e_0 \end{cases} \tag{4.13}$$

with T, e_0, k_1, and k_2 positive parameters. He analyzes the system in the (e, e') phase plane. The linear system on the region $|e| < e_0$ has a sink at $(0,0)$ and the other two systems have pseudo sinks at $(0,0)$. Kalman assumes that the first sink is real and the second two are complex, which amounts to declaring $k_1 t < 1/4$ and $k_2 t > 1/4$. He also declares $k_2 > k_1$ to ensure that the system responds slowly to small errors, but quickly to large ones. He sketches several composite trajectories by patching together partial trajectories of the linear systems (Figure 4.14). They appear to spiral toward the origin for a while, then approach it asymptotically. The transition graph that PLR constructs (Figure 4.15) proves that all trajectories follow this path. PLR can also sketch individual trajectories, as I will explain in Section 8.3. It handles Kalman's remaining examples analogously to the first one.

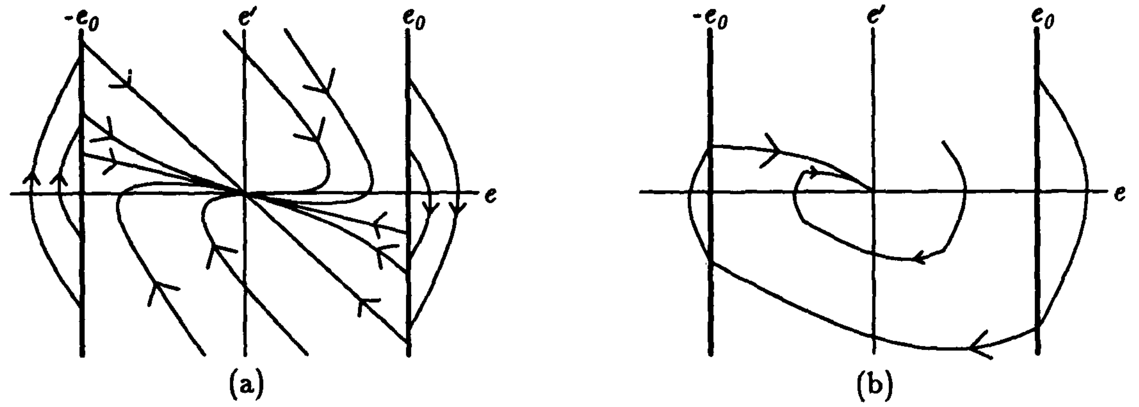


Figure 4.14: Trajectories for equation (4.12) drawn by Kalman (a) partial trajectories of linear systems (b) composite trajectories

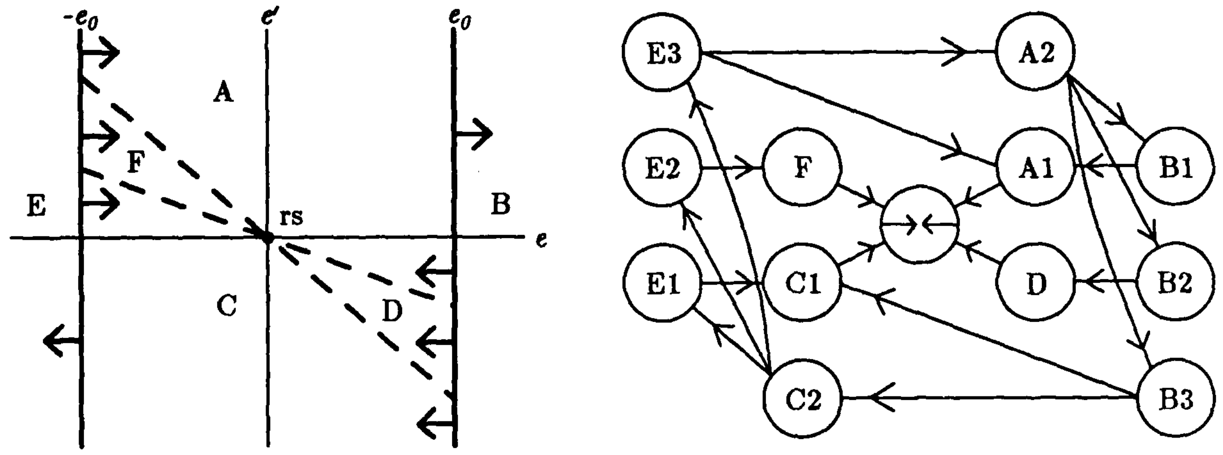


Figure 4.15: PLR's phase diagram and transition graph for equation (4.12)

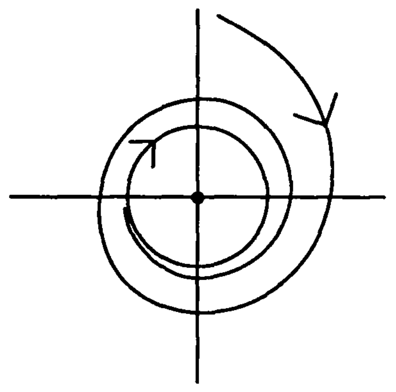
Chapter 5

Global Analysis

The examples in the previous chapter demonstrate the strengths and weaknesses of piecewise analysis. Although a transition graph for a system of differential equations captures the sequences of regions that trajectories traverse, it can miss some global properties. PLR addresses these lacunae with theoretical analysis that provides definitive information about the global behavior of many differential equations, including the examples from the previous chapter.

5.1 The Poincaré-Bendixson Theorem

PLR classifies systems of differential equations by the asymptotic behavior of their trajectories. The first three classes are trajectories that approach a fixed point, unbounded trajectories, and periodic trajectories. The Poincaré-Bendixson theorem [28, Ch. 10] states that all other trajectories of a two-dimensional system approach either a periodic trajectory, called a *limit cycle*, or a set of trajectories connecting fixed points, called a *separatrix*. Figure 5.1 demonstrates these possibilities. The remainder of this chapter describes propositions with which PLR determines the asymptotic behaviors that occur in particular systems. I demonstrates these propositions on the examples from Chapter 4. In the next chapter, I explain how PLR interprets transition graphs according to asymptotic behavior.



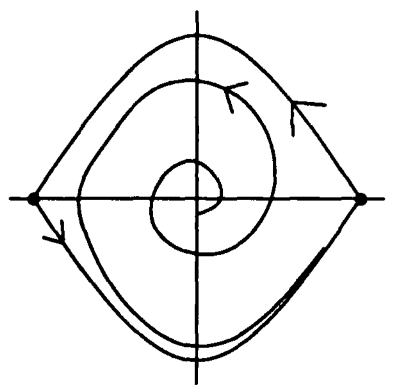


Figure 5.1: Asymptotic trajectories that approach a closed curve. All other bounded trajectories of a second-order system approach a fixed point by the Poincaré-Bendixson theorem

5.2 Liapunov Functions

The first few methods for determining asymptotic behavior involve the physical principle of conservation of energy. Every state of a system has a certain energy, a nonnegative real number. This quantity is constant along trajectories of *conservative* systems and nonincreasing along trajectories of *dissipative* systems. For example, the total energy of a mechanical system is the sum of the kinetic and potential energies of its components. The gravitational pendulum described in Section 4.2 has a single component, the bob, with kinetic energy $m(l\omega)^2/2$, potential energy $mgl(1-\cos\theta)$, and total energy

$$E(\theta,\omega) = \frac{m(l\omega)^2}{2} + mgl(1-\cos\theta). \tag{5.1}$$

The system is conservative in the undamped case ($\mu = 0$) and dissipative in the damped case ($\mu > 0$).

The energy function determines the asymptotic behavior of trajectories in conservative and dissipative systems. The trajectories of a conservative system lie on the level curves of the energy function. The level curves of the undamped pendulum are $E(\theta,\omega) = c$ with c a positive constant, or

$$\omega = \pm\sqrt{\frac{2}{l}\left(\frac{c}{lm} - g(1-\cos\theta)\right)} \tag{5.2}$$

in explicit form. The two components form a single periodic trajectory for $c < 2mgl$, a separatrix for $c = 2mgl$, and two unbounded trajectories for $c > 2mgl$ (Figure 5.2). The open curves map into closed curves in cylindrical space because they have the same value at both end points. In physical terms, the pendulum oscillates, approaches $(\pm\pi, 0)$, or spins depending on whether its total energy is less than, equal to, or greater than the potential energy at the maximum elevation $\theta = \pm\pi$.

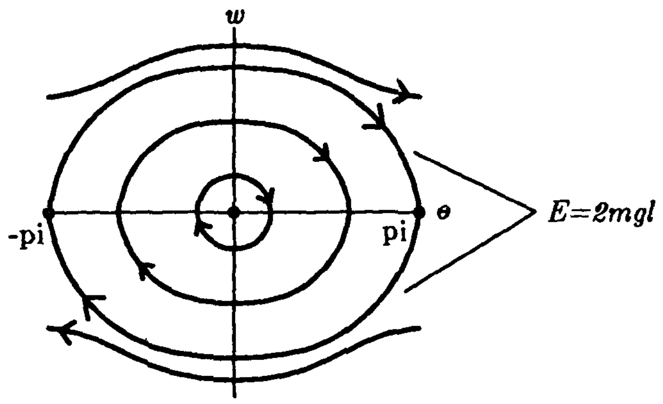


Figure 5.2: Level curves for the undamped pendulum

The trajectories of a dissipative system cannot ascend energy levels. They either remain on some energy level where dissipation does not occur or approach a *stationary point* of the energy function with respect to the system. For example, the fixed points of the damped pendulum are the only stationary points of its total energy. Every trajectory approaches one of these points asymptotically. The system has no dissipation-free energy levels because friction acts at all points where the pendulum moves, which is everywhere except at fixed points.

The discussion of energy functions applies to any nonnegative differentiable function $V : R^n \to R$ that is nonincreasing along trajectories of a system of differential equations $z' = f(\cdot)$. These are called *Liapunov functions* after their inventor. Define $V_i = \partial V/\partial z_i$. The function V is nonincreasing along trajectories iff the quantity

$$\dot{V}(z) = \text{grad}\, V(z) \cdot f(z) = \sum_{i=1}^{n} V_i(z) f_i(z) \tag{5.3}$$

is nonpositive, since each trajectory $\phi(t)$ satisfies

$$V'(\phi(t)) = \sum_{i=1}^{n} V_i(\phi(t))\phi_i'(t) = \sum_{i=1}^{n} V_i(\phi(t)) f_i(\phi(t)) = \dot{V}(\phi(t)), \tag{5.4}$$

by the chain rule and the definition of a solution. V is called a strict Liapunov function when $\dot{V} < 0$.

A fixed point $\bar{z}$ of a system $z' = f(z)$ is stable when there exists a neighborhood U of $\bar{z}$ and a differentiable function V such that

1. $V(\bar{z}) = 0$ and $V(z) > 0$ for $z \in U - \bar{z}$

2. $\dot{V} \leq 0$ on $U - \bar{z}$.

If $\dot{V} < 0$ in condition 2, $\bar{z}$ is asymptotically stable. These theorems, proved by Liapunov, formalize my intuitive discussion of dissipative systems. The quantity V represents energy; the condition $\dot{V} \leq 0$ expresses dissipation. The equation $V(z) = c$ represents a surface for $c > 0$. By condition 1, the surface has a component that contains $\bar{z}$ for c sufficiently small and shrinks to $\bar{z}$ as c approaches zero. By condition 2 and equation (5.3), the tangent to a trajectory on the boundary of a component points into the component or along the boundary. This implies that a trajectory remains in every component it enters. The strict version of condition 2 asserts that the tangent points into the component, implying that a trajectory crosses into the interior of every component it enters. This argument, illustrated in Figure 5.3, establishes Liapunov's theorem informally. Hirsch and Smale [19, Ch. 9] provide a rigorous proof.

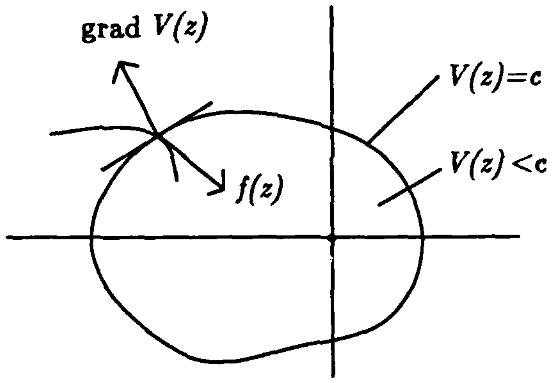


Figure 5.3: Trajectory pointing into a component

The total energy, E, is a Liapunov function for the undamped pendulum and a strict Liapunov function for the damped pendulum. It is nonnegative, differentiable, and the system derivative

$$\dot{E}(\theta, \omega) = -\mu(l\omega)^2 \tag{5.5}$$

is identically zero for $\mu = 0$ and negative for $\mu > 0$. The fixed point $(0,0)$ is stable for $\mu = 0$ and asymptotically stable for $\mu > 0$, by Liapunov's theorems. The fixed point $(\pm\pi, 0)$ is unstable because it has a positive eigenvalue.

The concept of total energy generalizes to systems of the form

$$\begin{cases} x' = y \\ y' = -g(x)y - h(x) \end{cases} \tag{5.6}$$

that consist of a dissipative force g *and a restoring force* h *that are functions only of position.* Every example in the previous chapter belongs to this category. The state variables x and y represent the position and velocity of an abstract particle of unit mass. The potential energy of the system equals the integral of the restoring force over displacement $H(x) = \int_0^x h(s)\,ds$ and the kinetic energy equals $y^2/2$. The total energy

$$V_e(x,y) = H(x) + \frac{y^2}{2} \tag{5.7}$$

has system derivative

$$\dot{V}_e(x,y) = -g(x)y^2. \tag{5.8}$$

V_e is a Liapunov function when H is nonnegative and g is nonpositive. PLR tests these conditions with BOUNDER. If they hold, it prunes paths in the transition graph along which V_e increases. For example, it prunes the path C-D3-E-I3-J in Figure 4.13 because the maximum energy in C is less than the minimum energy in J.

The following propositions about Liapunov functions justify my informal discussion of the pendulum's asymptotic behavior. The proofs appear in Brauer and Nohel [8, Ch. 5]. PLR applies these propositions to systems (5.6) for which it proves V_e Liapunov. Automatic construction of other Liapunov functions is a topic for further research. PLR can, however, apply any function that users provide.

Proposition 5.1 *Let V be a Liapunov function for a system. If $\dot{V} \equiv 0$, each trajectory lies on a surface $V(z) = c$ for some $c > 0$.*

This proposition holds for a Liapunov function V_e when $g \equiv 0$. The equation $V_e = c$ has two solutions

$$y = \pm\sqrt{2(c - H(x))} \tag{5.9}$$

on each interval of x values satisfying $c \geq H(x)$. The two solutions can form a closed curve, an open curve, or two curves (Figure 5.4). In the second and third cases, the domains of the curves can be finite, semi-infinite, or infinite. Each closed curve is a separatrix when $h(x) = 0$ at either of its zeroes and a periodic trajectory otherwise. An open curve is two trajectories when $h(x) = 0$ at its zero and a single trajectory otherwise. Trajectories never spiral toward a limit cycle or separatrix.

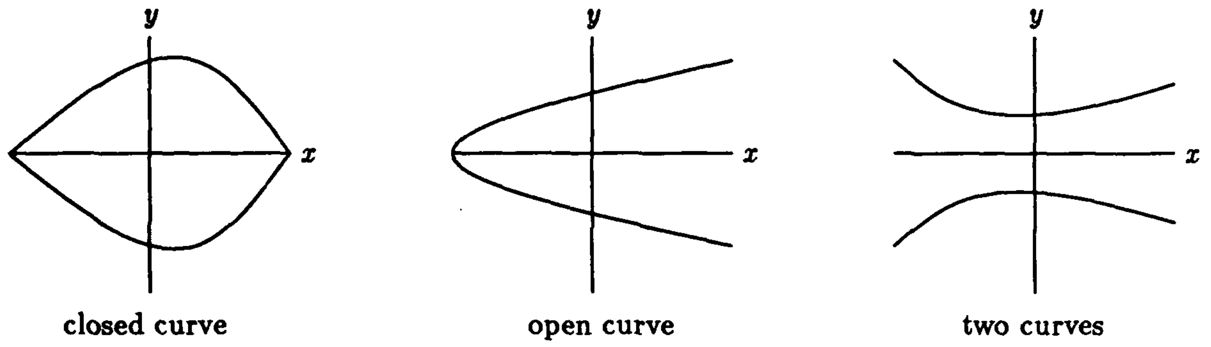


Figure 5.4: Components of $V_e(x, y) = c$

Piecewise analysis fails to discover that every trajectory of the undamped gravitational pendulum equation is periodic except for a separatrix between the oscillating and spinning trajectories. PLR proves this by applying Proposition 5.1 to E. The same proof works for the undamped magnetic pendulum equation (4.9). The proposition applies to the undamped horseshoe magnet equation (4.11), as well. PLR infers that the cycles in its transition graph (Figure 4.13) represent periodic trajectories.

Proposition 5.2 *Let a system have a Liapunov function V satisfying:* $\lim_{\|z\| \to \infty} V(z) = \infty$. *All trajectories are bounded.*

The condition implies that every level curve is bounded. Every trajectory lies in the closed region bounded by the initial energy curve, as discussed above. This implies the result. In a cylindrical phase space, a sequence $\|(\theta_i, \omega_i)\|$ approaches ∞ iff the sequence $|\omega_i|$ does, since $|\theta| \leq 2\pi$. Hence, the total energy, E, of the damped gravitational pendulum satisfies the precondition of Proposition 5.2, implying that all trajectories are bounded. The proposition also holds for simple and horseshoe damped magnetic pendulums.

Proposition 5.3 *Let a system have a Liapunov function V with $V(0) = 0$. Define $S = \{z|\dot{V}(z) = 0\}$. Let M be the largest invariant subset of S. All bounded trajectories approach M asymptotically.*

An invariant set is one that trajectories never leave after entering. Energy dissipates along every trajectory outside S. Trajectories approach S as their energy dies away. They remain near M by its definition. For equation (5.6), S is

$$\{(x,y)|g(x) = 0 \vee y = 0\}. \tag{5.10}$$

A point $(\bar{x},\bar{y})$ in S such that $\bar{y} \neq 0$ cannot be in M. The analyticity of g and the requirement that $g(\bar{x}) = 0$ imply that $g(x) \neq 0$ in some neighborhood n of $\bar{x}$. The trajectory through $(\bar{x},\bar{y})$ is strictly monotonic in x at $\bar{x}$ because $x' = y$. It immediately enters n, thus leaving S. This means that $(\bar{x},\bar{y}) \notin M$. A point $(\bar{x},0)$ in S is in M iff $h(\bar{x}) = 0$ by a similar argument. PLR finds these points with QMR (Sec. 8.1).

The set M for the total energy of the damped gravitational pendulum or damped simple magnetic pendulum consists of the fixed points: $(0,0)$ and $(\pm\pi,0)$. Propositions 5.2 and 5.3 imply that every trajectory approaches one of these points asymptotically. PLR infers that the infinite walks in the corresponding transition graphs (Figures 4.7 and 4.8) are vacuous. It cannot apply Proposition 5.3 to the damped horseshoe pendulum because QMR cannot solve $\dot{V}(z) = 0$ for that system.

5.3 Other Methods

The remaining methods for determining asymptotic behavior implement the following propositions whose proofs appear in Chapter 11 of Lefschetz [28].

Proposition 5.4 *A system has no separatrices if its fixed points are all sinks or all sources.*

Every separatrix contains a pair of (possibly equal) trajectories, u and v, and a fixed point, f, such that $\lim_{t\to\infty} u(t) = f$ and $\lim_{t\to-\infty} v(t) = f$. This cannot happen if the fixed points are all sinks or all sources. PLR rules out separatrices for the Lienard equation (2.3) and van der Pol equation (4.1) by this proposition. More sophisticated conditions, mainly involving index arguments, appear in Lefschetz.

Proposition 5.5 *Define the* divergence *of a two-dimensional system $z' = f(z)$ as*

$$\frac{\partial f_1}{\partial z_1} + \frac{\partial f_2}{\partial z_2}. \tag{5.11}$$

A closed set C in which the divergence is everywhere positive or everywhere negative (possibly with a finite number of exceptions) does not contain a periodic trajectory or a separatrix going from and to a single fixed point.

Suppose there exists a closed trajectory γ. The integral of the divergence along γ equals zero. The integral over the interior of γ has the same value by Green's theorem. But the integral cannot be zero because the divergence is everywhere positive or everywhere negative. This contradiction refutes the existence of γ. The divergence of the system (5.6) is $-g(x)$, so Proposition 5.5 holds iff g is almost everywhere positive or almost everywhere negative. The Lienard, gravitational pendulum, and magnetic pendulum equations satisfy the condition trivially with g a positive constant.

The last three propositions apply to (5.6) with g and h continuous and h Lipschitz. A function f is Lipschitz if $\exists k \forall x \forall y |f(x) - f(y)| < k|x - y|$.

Proposition 5.6 *Let $G(x) = \int_0^x g(s)\,ds$. Given*

1. *$g(x)$ is even and $g(0) < 0$,*

2. *$\lim_{x \to \infty} G(x) = \infty$,*

3. *G has a single positive zero: a,*

4. *$g(x)$ is positive for $x > a$, and*

5. *$h(x)$ is odd and $xh(x) > 0$ for $x \neq 0$.*

The system has a unique, orbitally stable limit cycle.

The first four conditions guarantee that the system gathers energy for small values of x and dissipates energy for large values. The final condition states that the restoring force always points inward. The result justifies my intuitive claim in Section 4.1 that all trajectories of the van der Pol equation approach a unique limit cycle. Trajectories outside the limit cycle spiral inward, whereas those inside spiral outward. Piecewise analysis determines that all trajectories spiral around the origin, but could not tell whether they move inward, move outward, or wobble around. PLR resolves the ambiguity with Proposition 5.6.

Proposition 5.7 *Let there exist positive numbers a, b, u, and v such that:*

1. *$g(x) \geq a$ for $|x| > u$, and*

2. *$h(x) \geq b$ for $x > v$ and $h(x) \leq -b$ for $x < -v$.*

There exists a compact set that all trajectories enter and never leave.

Intuitively, trajectories must eventually stop expanding if, for large enough displacements, g dissipates energy and h points inward. The van der Pol equation satisfies these conditions. Stronger results hold when g always dissipates energy and h increases monotonically with displacement.

Proposition 5.8 *Let there exist positive numbers a, b, c, and v such that:*

1. *$g(x) \geq a$,*

2. *$h(x) \geq b$ for $x > v$ and $h(x) \leq -b$ for $x < -v$,*

3. *$h'(x) \geq c$, and*

4. *$h''(x)$ exists and is bounded.*

All trajectories converge to one another.

PLR applies each of the propositions in this chapter to every system that it analyze . It tests the preconditions with the BOUNDER inequality prover (Ch. 7) and the QMR mathematical reasoner (Sec. 8.1). The next chapter describes how PLR uses the results.

Chapter 6

Transition Graph Interpretation

In this chapter, I explain how PLR compares the transition graphs for different piecewise linearizations of a system of differential equations. The goal is to determine whether one graph captures features of trajectories that the other misses. PLR uses this information to improve its initial piecewise linearization. It inserts additional linearization points until new features stop appearing in the resulting transition graphs. It can also compare externally specified piecewise linearizations with its own. PLR cannot compare transition graphs directly because each graph describes trajectories in terms of the regions defined by a specific piecewise linearization. It must derive region-independent descriptions of trajectories from the graphs and compare them.

PLR characterizes a transition graph by the qualitative features of its cycles and paths. It characterizes each cycle by the fixed points it encloses and by the possible asymptotic behaviors of its trajectories. Figure 6.1 lists the admissible behaviors. PLR determines which behaviors specific systems exhibit with the propositions described in the previous chapter. The number, location, and stability properties of limit cycles and of separatrices are not included in the description because these facts are hard or impossible to ascertain. Reasoning about them is a topic for future research.

For example, in the transition graph of the piecewise van der Pol equation for small values of k (Figure 4.3), the cycle A1-C-A2-B encloses the fixed point (0,0). Its trajectories approach a unique limit cycle, as shown in Proposition 5.6. PLR represents this with the ordered pair $\langle\{(0,0)\}, \{\text{limit-cycle}\}\rangle$, which does not express the uniqueness of the limit cycle. The cycle E2-H-C2-D in the transition graph for large values of k has the same description.

name	description
fixed-point	approaches a fixed point
unbounded	approaches infinity directly
unbounded-spiral	spirals toward infinity
periodic	is periodic
limit-cycle	is or approaches a limit cycle
separatrix	approaches a separatrix

Figure 6.1: Admissible asymptotic behaviors for a trajectory

PLR characterizes paths in a transition graph by the asymptotic behavior of their trajectories. It only considers maximal length paths, which represent complete trajectories. These trajectories subsume the partial trajectories of shorter paths. The final node of a maximal path characterizes its trajectories. If a path ends at the node for fixed point p, its trajectories approach p asymptotically. PLR represents this as the ordered pair $\langle\{p\}, \{\text{fixed-point}\}\rangle$.[1] If a path ends at a region node, the linear system for the region determines the asymptotic behavior of its trajectories. All such trajectories must directly approach one of the infinity points: $(-\infty,-\infty)$, $(-\infty,\infty)$, $(\infty,-\infty)$, or (∞,∞). They cannot approach a fixed point because the path does not end in a fixed point node. Nor can they spiral toward infinity without leaving the region. PLR invokes QMR (described in Chapter 8) to infer the infinity point, i, that trajectories approach and represents the result as $\langle\{i\}, \{\text{unbounded}\}\rangle$.

Figure 6.2 lists the abstract descriptions of the examples discussed in the previous chapters. Trajectories of the Lienard equation approach the origin along the path A-E1-(0,0) or approach $(-\infty,-\infty)$ along A-E3-C or D. All trajectories of the van der Pol equation approach a unique limit cycle for both small and large values of k. The trajectories of the undamped pendulum equation are periodic. Those on the paths D1-(0,0) and D4-C-D2-G enclose the origin, while those on F-D5-B and A-D3-E enclose no fixed points. In the damped case, all trajectories approach (0,0). The trajectories of the horseshoe pendulum are periodic, enclosing none, one, or both of the two fixed points p_1 and p_2. Although PLR determines the behavior of each example uniquely, that need not be the case in general.

PLR uses this abstraction to improve its initial approximations of systems. It bisects

[1]It omits unstable fixed points for simplicity because few or no trajectories approach them. They can be included if desired.

name	equation	description	Figures
Lienard	(2.3)	$\langle\{(0,0)\},\{\text{fixed-point}\}\rangle$, $\langle\{(-\infty,-\infty)\},\{\text{unbounded}\}\rangle$	3.8b
van der Pol	(4.1)	$\langle\{(0,0)\},\{\text{limit-cycle}\}\rangle$	4.3, 4.4
undamped pendulum	(1.2)	$\langle\{(0,0)\},\{\text{periodic}\}\rangle$,$\langle\{\},\{\text{periodic}\}\rangle$	4.6
damped pendulum	(4.4)	$\langle\{(0,0)\},\{\text{fixed-point}\}\rangle$	4.7, 4.8
horseshoe pendulum	(4.11)	$\langle\{\},\{\text{periodic}\}\rangle$,$\langle\{p_1\},\{\text{periodic}\}\rangle$, $\langle\{p_2\},\{\text{periodic}\}\rangle$,$\langle\{p1,p2\},\{\text{periodic}\}\rangle$	4.13

Figure 6.2: Qualitative features of systems

every finite interval in the approximation, analyzes the resulting system, and compares the abstract description of its transition graph to that of the original transition graph. It repeats this process until the new abstract description equals the old one. (This termination criterion is motivated by numeric integration algorithms that refine partitions until their intermediate results change less than some tolerance.) For example, Figure 6.3 shows how PLR refines its initial approximation of the Lienard equation (2.3). The transition graph for the refined system (Figure 6.4) is acyclic with sink nodes D, G, and (0,0). Trajectories that remain in D or G approach $(-\infty,-\infty)$; the rest approach (0,0). The description of the refined graph is identical to that of the original, listed in Figure 6.2, so refinement ends. Figure 6.5 demonstrates the close resemblance between the phase diagram of the actual Lienard equation and that of the original piecewise linear approximation. I explain how PLR produces phase diagrams in Section 8.3.

The motivation behind refinement is that finer approximations lead to more accurate predictions because they more closely resemble the actual equations. In fact, the sequence of refinements converges to the actual behavior of any individual system, but need not converge uniformly over all instances of a parameterized system. The refinement algorithm seeks the simplest approximation that captures the qualitative features of the original for all parameter values. Following qualitative analysis, numeric analysis can determine an approximation that reproduces the trajectories of the original for specific parameter values to within a prespecified numeric tolerance. This approximation will contain orders of magnitude more intervals than the qualitative one, enough to make piecewise analysis intractable.

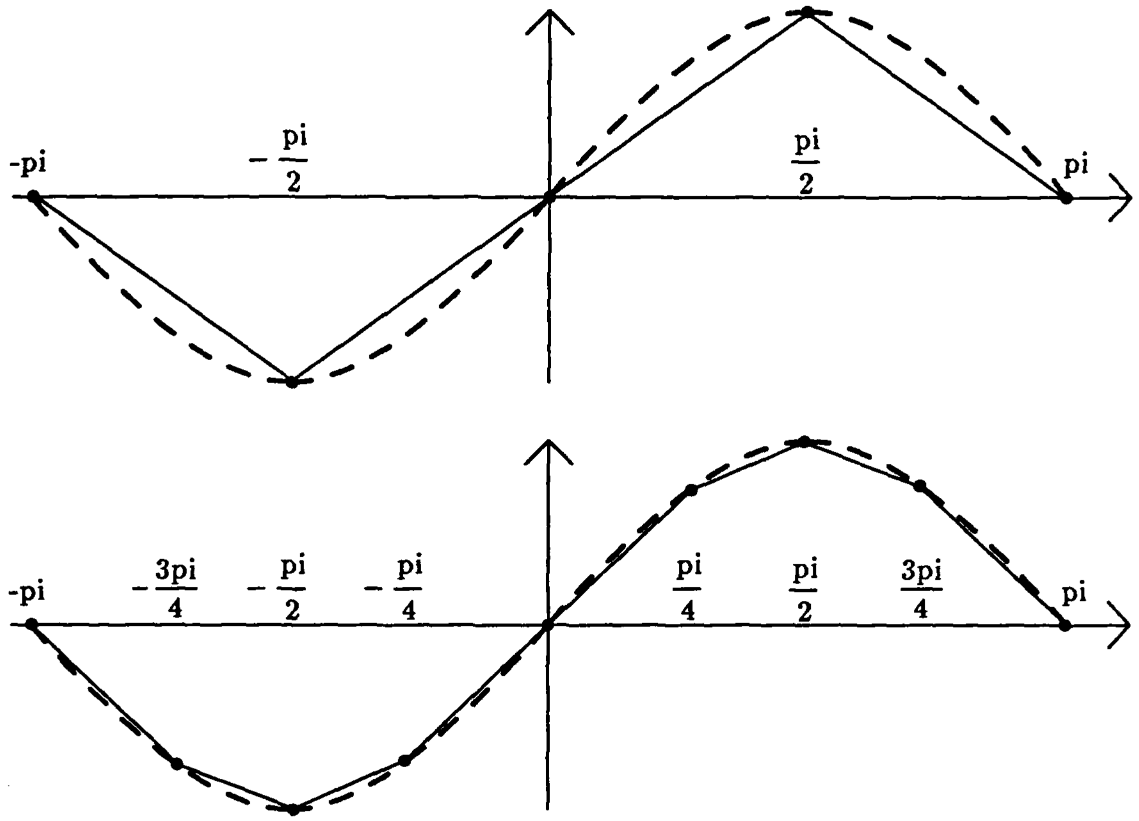


Figure 6.3: Refinement of the Lienard equation (2.3): initial approximation above and refined version below.

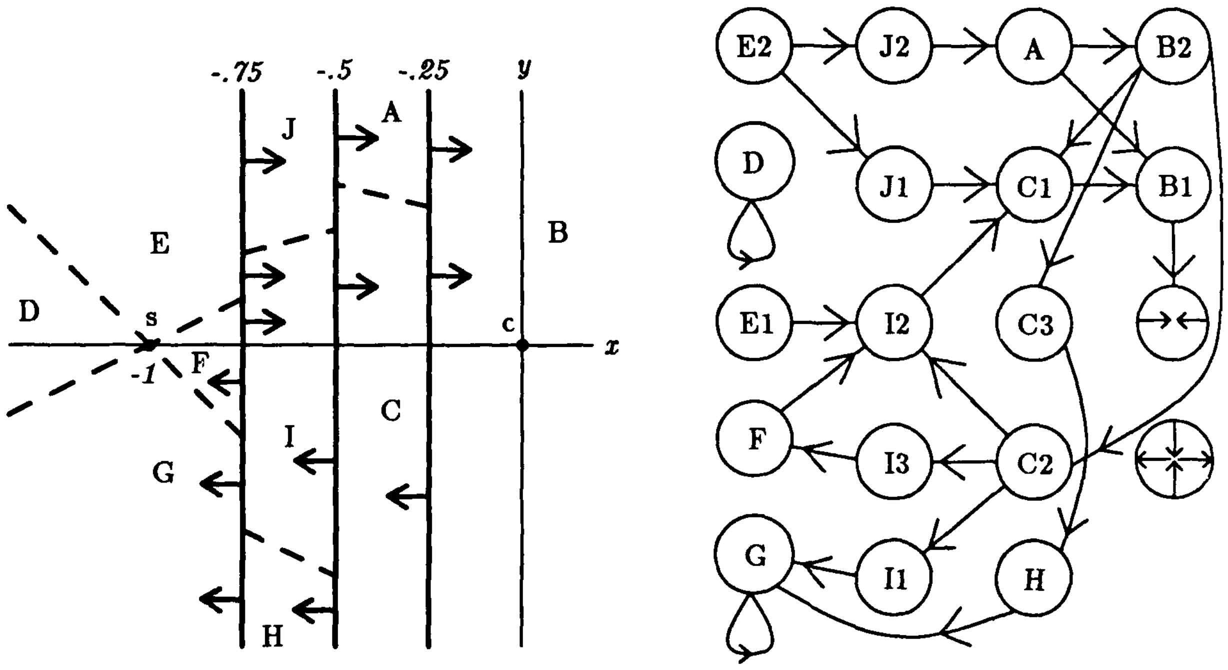


Figure 6.4: Phase diagram and transition graph for the refined Lienard equation

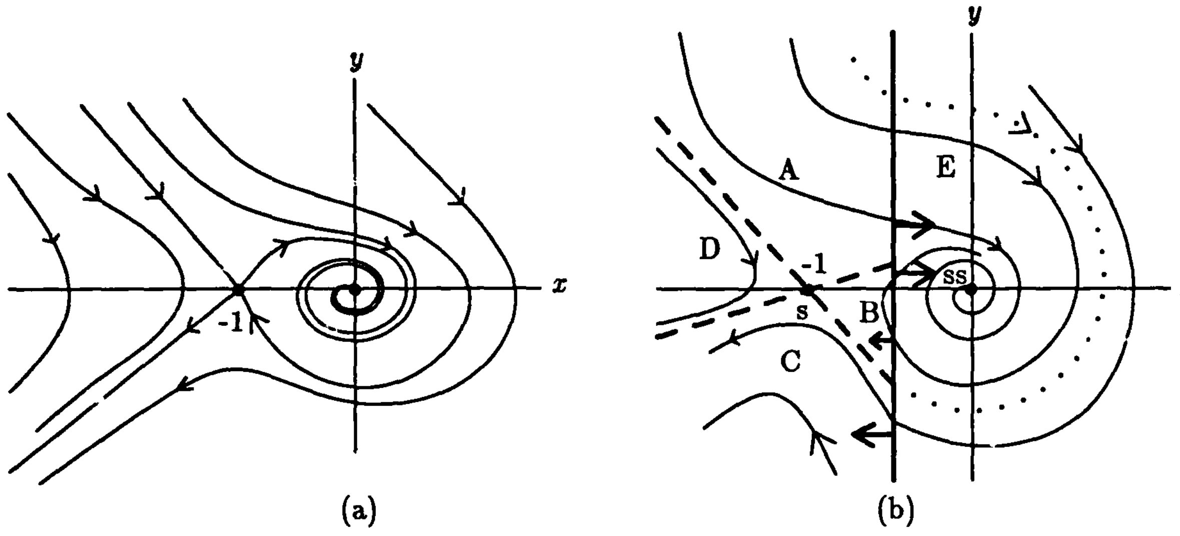


Figure 6.5: Phase diagrams of (a) the Lienard equation (b) its initial approximation

I chose a bisection refinement strategy for its simplicity and uniform applicability, but other strategies can easily be substituted. One might split intervals at inflection points and zeros of higher derivatives in addition to the extrema and zeroes of the original approximation. Another possibility is to split intervals where the derivative changes rapidly into more pieces than one splits intervals where it changes slowly.

This chapter concludes the description of the major components of PLR: construction of initial piecewise linear approximations, piecewise analysis, global analysis, transition graph interpretation, and refinement. The next two chapters describe PLR's principal utilities: the BOUNDER inequality prover and the QMR mathematical reasoner.

Chapter 7

Inequality Reasoning

7.1 Introduction

This chapter describes a program called BOUNDER that proves inequalities between functions over all points satisfying a finite set of constraints: equalities and inequalities between functions. I have designed BOUNDER for the purpose of efficiently resolving inequalities that arise in PLR and other realistic modeling problems. It handles universally quantified inequalities, which make up the majority of these problems, but ignores the complexities of arbitrary quantification. BOUNDER manipulates *extended elementary functions:* polynomials and compositions of exponentials, logarithms, trigonometric functions, inverse trigonometric functions, absolute values, maxima, and minima. It cannot prove every true inequality in its domain, since Richardson [36] proves this problem undecidable. It can, however, resolve all the inequalities that arise in this thesis and many more. Inequality provers for decidable subsets of the extended elementary functions, such as the linear functions, do not meet the needs of PLR or of my other research.

BOUNDER tests whether a set of constraints, S, implies an inequality $a \leq b$ by calculating upper and lower bounds for $a - b$ over all points satisfying S. It proves the inequality when the upper bound is negative or zero, refutes it when the lower bound is positive, and fails otherwise. Specific bounding methods perform well on some subset of the extended elementary functions, but poorly elsewhere, so BOUNDER maintains a hierarchy of increasingly complex methods. When one fails to resolve an inequality, it tries the next. Although complex methods derive tighter bounds than simple ones for most functions, exceptions exist.

Hence, the hierarchy derives tighter bounds than even its most powerful component. It also performs well on hard problems without wasting time on easier ones.

BOUNDER consists of a simplifier, a context manager, and four bounding algorithms: bounds propagation, substitution, derivative inspection, and iterative approximation. The simplifier reduces input inequalities to equivalent but shorter inequalities by canceling common terms and replacing monotonic functions with their arguments. For example, $x + 1 \leq y + 1$ simplifies to $x \leq y$, $-x \leq -y$ to $x \geq y$, $x^3 \leq y^3$ to $x \leq y$, and $e^x \leq e^y$ to $x \leq y$. The simplifier only cancels multiplicands whose signs it can determine by bounds propagation. The context manager organizes constraint sets in the format required by the bounding algorithms. The bounding algorithms derive upper and lower bounds for a function over all points satisfying a constraint set. In the remainder of this chapter, I describe these components, compare BOUNDER with other inequality reasoners, and draw conclusions.

BOUNDER distinguishes between strict and non-strict inequalities. This mechanism consists mainly of careful bookkeeping, based on the properties of continuous functions. For instance, a sum can only attain its upper or lower bound if both addends can attain theirs. From here on, I will speak only of non-strict inequalities, since BOUNDER handles the strict case analogously.

7.2 The Context Manager

The context manager (CM) manages constraint sets for the other components. The simplest constraints, called *relational constraints*, are equalities and inequalities between extended elementary functions. CM derives an upper or lower bound for a variable x from an inequality $L \leq R$ by reformulating it as $x \leq U$ or $x \geq U$ with U free of x. It derives upper and lower bounds for x from an equality $L = R$ by reformulating it as $x = U$. Inequality manipulation may depend on the signs of the expressions involved. For example, the constraint $ax \leq b$ can imply $x \leq b/a$ or $x \geq b/a$ depending on the sign of a. In such cases, CM attempts to derive the relevant signs from other members of the constraint set using bounds propagation. If it fails, it ignores the constraint. (Another possibility would be to create a disjunctive constraint, explained below.) Constraints whose variables cannot be isolated, such as $x \leq 2^x$, are ignored as well. CM gathers the bounds that it derives from a set of relational constraints into a *simple context.* The number of variables in a constraint is linear in its length and each variable requires linear time to isolate. Isolation may require deriving the signs of all the subexpressions in the constraint. Theorem 1 implies that this process takes linear time. All told, processing each constraint requires quadratic time in its length. Subsequent complexity results exclude this time.

The context manager also handles ***propositional constraints,*** boolean combinations of relational constraints. A set of propositional constraints is equivalent to the conjunction of its members. CM eliminates negation from this conjunction by replacing every instance of $a \neq b$ with $a < b \vee b < a$, every instance of $\neg(a \leq b)$ with $b < a$, and so on. It recasts the resulting formula in disjunctive normal form as a disjunction of conjunctions of relational constraints. CM creates a simple context for each disjunct and collects the results into a *disjunctive context.*

The bounding algorithms and the inequality prover reduce disjunctive contexts to simple ones. The upper bound of a function in a disjunctive context is the maximum of its upper bounds in all disjuncts and the lower bound is the minimum of its lower bounds. For example, the constraint $x \leq -1 \vee x \geq 2$ implies a lower bound of 1 for x^2. Similarly, an inequality holds in a disjunctive context iff it holds in all disjuncts. The remainder of this chapter deals solely with relational constraints and simple contexts, hereafter abbreviated to constraints and contexts.

Two pairs of functions form the interface between the context manager and the inequality prover and bounding algorithms. Given a variable x and a constraint set S, the functions VAR-LB$_S(x)$ and VAR-UB$_S(x)$ return the maximum of x's numeric lower bounds in S and the minimum of its numeric upper bounds. The functions LOWER$_S(x)$ and UPPER$_S(x)$ return the maximum over all lower bounds, both symbolic and numeric, and the minimum over all upper bounds. Both VAR-LB and LOWER derive lower bounds for x, whereas both VAR-UB and UPPER derive upper bounds. However, LOWER and UPPER produce tighter bounds then VAR-LB and VAR-UB because they take symbolic constraints into account. Figure 7.1 demonstrates the respective bounds that these functions derive. All four functions run in constant time once the contexts are constructed.

	numeric constraints		all constraints	
x	VAR-LB$_S(x)$	VAR-UB$_S(x)$	LOWER$_S(x)$	UPPER$_S(x)$
a	1	∞	1	$-4/b$
b	-2	0	$\max\{-2, -4/a, c\}$	$\min\{0, c\}$
c	$-\infty$	∞	b	b

Figure 7.1: Numeric and general bounds for $S = \{a \geq 1,\ b \leq 0,\ b \geq -2,\ ab \geq -4,\ c = b\}$

7.3 Bounding Algorithms

This section contains the details of the four bounding algorithms: bounds propagation, substitution, derivative inspection, and iterative approximation. Each algorithm derives tighter bounds than its predecessor, but takes more time. Each invokes all of its predecessors for subtasks, except that derivative inspection never calls bounds propagation. The bounding algorithms define the extended elementary functions on the extended real numbers in the standard fashion, that is $1/\pm\infty = 0$, $\log 0 = -\infty$, $2^\infty = \infty$, and so on. Throughout this chapter, "number" refers to an extended real number.

7.3.1 Bounds Propagation

The bounds propagation algorithm bounds a compound function by bounding its components recursively and combining the results. For example, the upper bound of a sum is the sum of the upper bounds of its addends. The recursion terminates when it reaches numbers and variables. Numbers are their own bounds; VAR-LB and VAR-UB bound variables. Figure 7.2 contains the bounds propagation algorithm, BP$_S(e)$, for a function e over a set of constraints S. One can represent e as an expression in its variables $x_1, \ldots, x_n$ or as a function $e(x)$ of the vector $x = (x_1, \ldots, x_n)$. From here on, these forms are used interchangeably. The notations lb_e and ub_e abbreviate the lower and upper bounds LB$_S(e)$ and UB$_S(e)$. Figure 7.3 contains the subroutine that bounds exponentials. Figure 7.4 shows the upper bound subroutine for trigonometric functions. The lower bounds are obtained by replacing max with min, ∞ with $-\infty$, TRIG-UB with TRIG-LB, and line 1.1 with

1.1′ $\exists n \in \mathcal{Z}\ lb_a < (2n - \frac{1}{2})\pi < ub_a \quad -1$

	e is	lb_e	ub_e
1	a number	e	e
2	a variable	VAR-LB$_S(e)$	VAR-UB$_S(e)$
3	$a+b$	$lb_a + lb_b$	$ub_a + ub_b$
4	ab	$\min\{lb_a lb_b,\ lb_a ub_b,\ ub_a lb_b,\ ub_a ub_b\}$	$\max\{lb_a lb_b,\ lb_a ub_b,\ ub_a lb_b,\ ub_a ub_b\}$
5	a^b	EXPT-LB$_S(a,b)$	EXPT-UB$_S(a,b)$
6	$\min\{a,b\}$	$\min\{lb_a, lb_b\}$	$\min\{ub_a, ub_b\}$
7	$\max\{a,b\}$	$\max\{lb_a, lb_b\}$	$\max\{ub_a, ub_b\}$
8	$\log a$	$\log lb_a$	$\log ub_a$
9	$\lvert a\rvert$		
	9.1 $lb_a < 0 < ub_a$	0	$\max\{-lb_a, ub_a\}$
	9.2 else	$\min\{\lvert lb_a\rvert, \lvert ub_a\rvert\}$	$\max\{\lvert lb_a\rvert, \lvert ub_a\rvert\}$
10	trigonometric	TRIG-LB$_S(e)$	TRIG-UB$_S(e)$

Figure 7.2: The BP$_S(e)$ Algorithm

	Case	EXPT-LB$_S(a,b)$	EXPT-UB$_S(a,b)$
1	$lb_a > 0$	$\exp(lb_{b\log a})$	$\exp(ub_{b\log a})$
2	$b = \frac{p}{q}$ with p,q integers		
	2.1 p,q odd and positive	$[lb_a]^b$	$[ub_a]^b$
	2.2 p,q odd and $ub_a < 0$	$[ub_a]^b$	$[lb_a]^b$
	2.3 p even	$\exp(lb_{b\log\lvert a\rvert})$	$\exp(ub_{b\log\lvert a\rvert})$
	2.4 else	$-\infty$	∞
3	else	$-\infty$	∞

Figure 7.3: Bounding Algorithms for Exponentials

	e is	TRIG-UB$_S(e)$
1	$\sin a$	
	1.1 $\exists n \in \mathcal{Z}\ lb_a < (2n+\frac{1}{2})\pi < ub_a$	1
	1.2 else	$\max\{\sin lb_a, \sin ub_a\}$
2	$\cos a$	TRIG-UB$_S(\sin(a+\frac{\pi}{2}))$
3	$\tan a$	
	3.1 $\exists n \in \mathcal{Z}\ lb_a < (n+\frac{1}{2})\pi < ub_a$	∞
	3.2 else	$\tan ub_a$
4	$\arcsin a$	
	4.1 $lb_a \geq -1 \wedge ub_a \leq 1$	$\arcsin ub_a$
	4.2 else	∞
5	$\arccos a$	
	5.1 $lb_a \geq -1 \wedge ub_a \leq 1$	$\arccos lb_a$
	5.2 else	∞
6	$\arctan a$	$\arctan ub_a$

Figure 7.4: Upper bounds for Trigonometric Functions

and by interchanging all instances of lb and ub that appear in the second column.

The correctness and complexity of BP are summarized in the theorem:

Theorem 1 *For any extended elementary function $e(x)$ and set of constraints S, bounds propagation derives numbers lb_e and ub_e satisfying*

$$\forall x.satisfies(x, S) \Rightarrow lb_e \leq e(x) \leq ub_e \tag{7.1}$$

in time proportional to e's length.

Proof: The proof is by induction on e's length. First, consider correctness. Numbers and variables are the only inputs of length 1. Numbers satisfy condition (7.1) identically and variables satisfy it by the definitions of VAR-LB and VAR-UB. Suppose condition (7.1) holds for inputs of length less than n and let e be of length n. BP bounds e by combining the bounds of its components, a and b, in one of steps 3–10 (Figure 7.2). The correctness of the bounds on a and b follows from the inductive hypothesis, since they have length less than n. It remains to verify that the combination rules yield valid bounds. Straightforward algebraic manipulations, performed in full by Moore [32], confirm steps 3,4,6,7 and 9. Step 8 follows from the fact that $\log x$ increases monotonically in x, with the provision that undefined LB values represent $-\infty$ and undefined UB values, ∞. The remaining steps, 5 and 10, call the exponential and trigonometric bounding algorithms respectively.

Step 1 of the exponential bounding algorithm (Figure 7.3) uses the monotonicity of the exponential function along with the identity $a^b = \exp(b \log a)$ for $a \geq 0$. The inductive hypothesis does not establish the correctness of the bounds on $b \log a$ because the expression has length n. I must apply the arguments for steps 4 and 8 of BP to the bounds of a and b. Steps 2.1 and 2.2 are valid because the function a^b respectively increases and decreases monotonically in a for those choices of b. In step 2.3 the equality $a^b = |a|^b$ holds, so the proof of step 1 applies. Finally, steps 2.4 and 3 yield valid bounds vacuously.

The correctness of the trigonometric bounding algorithms (Figure 7.4) follows from the properties of piecewise monotonic functions. Suppose an interval A partitions into a set of subintervals D on which a function f decreases monotonically and a set I on which f increases monotonically. The maximum (minimum) of f on A occurs at the right (left) endpoint of some interval in I or at the left (right) endpoint of some interval in D. When specialized to the individual trigonometric functions, this result implies the correctness of their bounds. This completes the correctness proof.

Next, consider the time-complexity of BP, $t(n)$. Inputs of length 1, numbers and variables, take constant time. A unary function of length n takes time $t(n-1)$ to calculate the bounds of its argument plus constant overhead. A binary function of length n takes time $t(i)+t(n-i-1)$ to calculate the bounds of its arguments, with i the length of its first argument, plus constant overhead. Induction proves that $t(n)$ is of order kn, with k the maximum overhead required for any step of BP. ∎

BP achieves linear time-complexity by ignoring constraints among variables or multiple occurrences of a variable in an expression. It derives excessively loose bounds when these factors prevent all the constituents of an expression from varying independently over their ranges. For instance, the constraint $a \leq b$ implies that $a - b$ cannot be positive. Yet given only this constraint, BP derives an upper bound of ∞ for $a - b$ by adding the upper bounds of a and $-b$, both ∞. As another example, when no constraints exist, the joint occurrence of x in the constituents of $x^2 + x$ implies a global minimum of $-1/4$. Yet BP deduces a lower bound of $-\infty$ by adding the lower bounds of x^2 and x, 0 and $-\infty$. Subsequent bounding algorithms derive optimal bounds for these examples. Substitution analyzes constraints among variables and the final two algorithms handle multiple occurrences of variables. All three obtain better results than BP, but pay an exponential time-complexity price.

7.3.2 Substitution

The su' 'itution algorithm constructs bounds for an expression by replacing some of its variables with their bounds in terms of the other variables. Substitution exploits all solvable constraints, whereas bounds propagation limits itself to constraints between variables and numbers. In our previous example, substitution derives an upper bound of 0 for $a - b$ from the constraint $a \leq b$ by bounding a from above with b, that is $a - b \leq b - b = 0$. Substitution is performed by the algorithms $\text{SUP}_S(e, H)$ and $\text{INF}_S(e, H)$, which calculate upper and lower bounds on e over the constraint set S in terms of the variable set H. When H is empty, the bounds reduce to numbers.

Figures 7.5 and 7.6 contain the SUP function and its auxiliary, SUPP. One obtains the INF function from Figure 7.5 by interchanging all occurrences of SUP and INF and replacing SUPP with INFF, UPPER with LOWER, EXPT-SUP with EXPT-INF, TRIG-SUP with TRIG-INF, and max with min. Also, step 9 is replaced with:

```
9'  $|a|$
    9.1  $\text{INF}_S(a,H) < 0 < \text{SUP}_S(a,H)$    $0$
    9.2  else                                           $\min\{|\text{INF}_S(a,H)|, |\text{SUP}_S(a,H)|\}$
```

The INFF function is obtained from Figure 7.6 by replacing SUPP with INFF and UB with LB. The auxiliary functions EXPT-SUP, EXPT-INF, TRIG-SUP, and TRIG-INF are derived from the exponential and trigonometric bounding algorithms (Figures 7.3 and 7.4) by replacing $\text{UB}_S(a)$ with $\text{SUP}_S(a, H)$, replacing $\text{LB}_S(a)$ with $\text{INF}_S(a, H)$, and so on for b. The expression $v(e)$ denotes the variables contained in e. In the remainder of this section, I will focus on SUP. INF is analogous.

```
     e is                          SUP_S(e,H)
1    $v(e) \subseteq H$            $e$
2    a variable                    $\text{SUPP}_S(e, \text{SUP}_S(\text{UPPER}_S(e), H \cup \{e\}))$
3    $a + b$
     3.1 $v(b) - v(a) \subseteq H$ $\text{SUP}_S(a,H) + \text{SUP}_S(b,H)$
     3.2 else                      $\text{SUP}_S(a + \text{sup}_S(b, H \cup v(a)), H)$
4    $ab$
     4.1 $\text{LB}_S(a) \geq 0$
         4.1.1 $v(b) - v(a) \subseteq H$   $\max\{\text{SUP}_S(a,H)\text{SUP}_S(b,H), \text{INF}_S(a,H)\text{SUP}_S(b,H)\}$
         4.1.2 else                        $\text{SUP}_S(a\,\text{SUP}_S(b, H \cup v(a)), H)$
     4.2 $\text{UB}_S(a) \leq 0$
         4.2.1 $v(b) - v(a) \subseteq H$   $\max\{\text{SUP}_S(a,H)\text{INF}_S(b,H), \text{INF}_S(a,H)\text{INF}_S(b,H)\}$
         4.2.2 else                        $\text{SUP}_S(a\,\text{INF}_S(b, H \cup v(a)), H)$
     4.3 else                      $\max\{\text{SUP}_S(a,H)\text{SUP}_S(b,H), \text{SUP}_S(a,H)\text{INF}_S(b,H),$
                                          $\text{INF}_S(a,H)\text{SUP}_S(b,H), \text{INF}_S(a,H)\text{INF}_S(b,H)\}$
5    $a^b$                         $\text{EXPT-SUP}_S(a,b,H)$
6    $\min\{a,b\}$                 $\min\{\text{SUP}_S(a,H), \text{SUP}_S(b,H)\}$
7    $\max\{a,b\}$                 $\max\{\text{SUP}_S(a,H), \text{SUP}_S(b,H)\}$
8    $\log a$                      $\log \text{SUP}_S(a,H)$
9    $|a|$                         $\max\{|\text{INF}_S(a,H)|, |\text{SUP}_S(a,H)|\}$
10   trigonometric                 $\text{TRIG-SUP}_S(e,H)$
```

Figure 7.5: The $\text{SUP}_S(e, H)$ Algorithm

	case	$\text{SUPP}_S(x, B)$
1	$x \notin v(B)$	B
2	$B = rx + A;\ r \in \Re,\ x \notin v(A)$	
	2.1 $r \geq 1$	∞
	2.2 $r < 1$	$\frac{A}{1-r}$
3	$B = \min\{C, D\}$	$\min\{\text{SUPP}_S(x, C), \text{SUPP}_S(x, D)\}$
4	$B = \max\{C, D\}$	$\max\{\text{SUPP}_S(x, C), \text{SUPP}_S(x, D)\}$
5	else	$\text{UB}_S(B)$

Figure 7.6: The $\text{SUPP}_S(x, B)$ Algorithm

In step 1, SUP calculates the upper bounds of numbers and of variables included in H. It analyzes a variable, x, not in H by constructing an intermediate bound

$$B = \text{SUP}_S(\text{UPPER}_S(x), H \cup \{x\}) \tag{7.2}$$

for x and calling SUPP to derive a final bound. If possible, SUPP derives an upper bound for x in H directly from the inequality $x \leq B$. Otherwise, it applies bounds propagation to B. For instance, the inequality $x \leq 1 - x$ yields an upper bound of $1/2$, but $x \leq x^2 - 1$ does not provide an upper bound, so SUPP returns $\text{UB}_S(x^2 - 1)$.

SUP exploits constraints among variables to improve its bounds on sums and products. If b contains variables that a lacks, but which have bounds in a's variables, SUP constructs an intermediate upper bound, U, for $a + b$ or ab by replacing b with these bounds. A recursive application of SUP to U produces a final upper bound. (Although not indicated explicitly in Figure 7.5, these steps are symmetric in a and b.) If a and b have the same variables, SUP bounds $a + b$ and ab by recursively bounding a and b and applying bounds propagation to the results. For example, given the constraints $c \geq 1$, $d \geq 1$, and $cd \leq 4$, SUP derives an interme''ate bound of $3c/4$ for $c - 1/d$ by replacing $-1/d$ with $-c/4$, its upper bound in c. This bound is derived as follows:

$$\text{SUP}(-\frac{1}{d}, \{c\}) = -\text{INF}(\frac{1}{d}, \{c\}) = -\frac{1}{\text{SUP}(d, \{c\})} = -\frac{1}{4/c} = -\frac{c}{4} \tag{7.3}$$

SUP uses the recursive call

$$\text{SUP}(\frac{3c}{4}, \{\}) = \text{SUPP}(c, \text{SUP}(\frac{3}{d}, \{c\})) = \text{SUPP}(c, \frac{3}{\text{INF}(d, \{c\})}) = \text{SUPP}(c, 3) = 3 \tag{7.4}$$

to derive a final bound of 3 for $c - 1/d$.

Substitution produces valid bounds for any input, variable set, and constraints. Although the proof requires a joint induction on both SUP and INF, I will present only the SUP half, since the INF half is completely analogous. At first glance, it seems possible that SUP fails to terminate on some inputs. Step 2 bounds a variable by invoking SUP recursively on a more complex expression, its UPPER. Similarly, steps 3.1, 4.1.2, and 4.2.2 bound their inputs by recursing on intermediate values of unknown complexity. The following lemma rules out infinite recursion and allows us to proceed to the correctness theorem:

Lemma 2 *The algorithms* $\text{SUP}_S(e,H)$ *and* $\text{INF}_S(e,H)$ *terminate for any input* e*, variable set* H*, and constraint set* S*.*

Proof: The auxiliary functions EXPT-SUP and TRIG-SUP terminate if SUP does, since they call it at most twice and apply a few extended elementary functions to the results. Regardless of SUP, SUPP always terminates because each recursive call, in steps 2 and 3, decreases the length of b, while the other steps terminate directly. It remains to show that SUP makes only finitely many recursive calls on any input. Let T be the invocation tree of recursive calls made by SUP for some e, H, and S. Each node n is labeled with the input e_n and variable set H_n of its corresponding SUP call. A node n *takes step* k if e_n matches case k of SUP (Figure 7.5). Let V denote the union of e's variables with those appearing in S. The variables of every e_n form a subset of V. Consider some path p through T (Figure 7.7). Each node l of p that takes step 2 adds some member of V to the set H_m of its successor m. No step removes elements from H, so this variable appears in all subsequent H_n. At most $|V|$ nodes of p can take step 2 before some node takes step 1 and terminates the path.

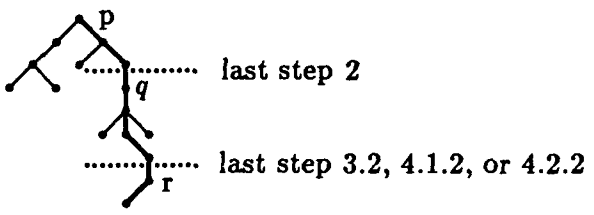


Figure 7.7: A path p through T

Let q be the sub-path of p beginning with the successor of the last node that takes step 2. By the argument above, p is finite iff q is finite. Let the *free set* of a node , denote the variables of e_n not in H_n. Suppose n's predecessor, m, takes step 3.2, 4.1.2, or 4.2.2 and n corresponds to the outer SUP of that step. The expression e_m has the form $a + b$ or ab with b containing some variable not in $v(a) \cup H_m$, whereas e_n contains only variables from $v(a) \cup H_n$. This implies that n's free set is a proper subset of m's, since H_m equals H_n. None of SUP's steps except 2 increases the free set of a node over that of its predecessor. Path q contains no instances of step 2, so it can contain only finitely many such successors of nodes that take steps 3.2, 4.1.2, and 4.2.2 before it reaches a node with an empty free set and terminates at step 1. Let r be the sub-path of q beginning with the successor of the last node that takes one of these three steps. Each of r's nodes takes a step that reduces the size of its input, so r must terminate. This proves q, and hence p, finite. ∎

The following theorem establishes the correctness of substitution:

Theorem 3 *For every extended elementary function e, variable set H, and constraint set S, the expressions $i = \text{INF}_S(e, H)$ and $s = \text{SUP}_S(e, H)$ satisfy the conditions:*

$$i \text{ and } s \text{ are expressions in } H \tag{7.5}$$

$$\forall x.\mathit{satisfies}(x, S) \Rightarrow i(x) \leq e(x) \leq s(x) \tag{7.6}$$

Proof: I will prove that every finite invocation tree for SUP satisfies conditions (7.5) and (7.6) by induction on tree depth. This proves the theorem, since both algorithms always produce finite trees by Lemma 2. Trees of depth 1 return bounds of e, which satisfy condition (7.5) trivially and (7.6) identically. Suppose the theorem holds for trees of depth less than n and consider a SUP tree T of depth $n > 1$. The correctness of all recursive calls follows from the inductive hypothesis. It remains to prove that the root node satisfies both conditions.

Step 1 of SUP cannot occur since n is greater than 1. By inductive hypothesis and by the definition of the UPPER function, the second argument to SUPP in step 2 of SUP is an expression in $H \cup \{\epsilon\}$ that bounds e. This implies directly that step 1 of SUPP satisfies conditions (7.5) and (7.6). Steps 2.1, 2.2, and 2.3 satisfy them vacuously, by elementary algebra, and by Theorem 1 respectively. Since the other steps satisfy the conditions, the definitions of min and max guarantee that steps 3 and 4 do too.

Steps 3.2, 4.1.2, and 4.2.2 of SUP generate an intermediate upper bound, u, by bounding one of e's constituents. Their final result, $\text{SUP}(u, H)$, satisfies conditions 7.5 and 7.6 by the inductive hypothesis. The validity of u follows from the facts

$$\begin{array}{lrcl} \text{(step 3.2)} & k \leq l & \Rightarrow & a+k \leq a+l \\ \text{(step 4.1.2)} & a \geq 0 \wedge k \leq l & \Rightarrow & ak \leq al \\ \text{(step 4.2.2)} & a \leq 0 \wedge k \geq l & \Rightarrow & ak \leq al \end{array}$$

by the inductive hypothesis. SUP's remaining steps, including the auxiliary functions EXPT-SUP and TRIG-SUP, apply the combination rules of BP, appearing in Figures 7.2–7.4, to the SUP and INF of e's constituents. Their results satisfy condition (7.5) because the constituents do. Condition (7.6) holds by Theorem 1. ∎

Substitution utilizes constraints among variables to improve on the bounds of BP, but ignores constraints among multiple occurrences of variables. It performs identically to BP on the example of $x^2 + x$, deriving a lower bound of $-\infty$. Yet that bound is overly pessimistic because no value of x minimizes both addends simultaneously. The last two bounding algorithms address this shortcoming.

7.3.3 Derivative Inspection

The derivative inspection algorithm, DI, calculates bounds for a function f from the signs of its partial derivatives. If the partial derivative of f with respect to x is non-negative over an interval $[l, r]$, then f's minimum and maximum on the interval, for any choice of its other variables, occur at l and r respectively. If the partial derivative is non-positive, the maximum occurs at l and the minimum at r. In the example from the previous section, the derivative of $x^2 + x$ is non-positive on $[-\infty, -1/2]$ and non-negative on $[-1/2, \infty]$. This information enables DI to derive an optimal lower bound of $-1/4$ for $x^2 + x$, as opposed to INF's bound of $-\infty$.

Before describing DI in detail, I will introduce some notation. Let S be a set of constraints and x a vector $(x_1, \ldots x_n)$ of variables. The *range* of x_i in S is the interval

$$X_i = [\text{INF}_S(x_i, \{\}), \text{SUP}_S(x_i, \{\})] \tag{7.7}$$

and the *range* of x in S is the Cartesian product $X = X_1 \times \cdots \times X_n$ of its components' ranges. Theorem 3 implies that

$$\forall x. \textit{satisfies}(x, S) \Rightarrow x \in X \tag{7.8}$$

One can split X in dimension i by choosing a point p_i in X_i and forming two subsets, one with the additional constraint $x_i \leq p_i$ and the other with $x_i \geq p_i$. One can collapse X to a point in dimension i by adding the constraint $x_i = p_i$ to S. If X is collapsed in all directions, it reduces to a point.

DI bounds a function $f(x)$ by partitioning X into subregions on which $f(x)$ is monotonic. The maximum upper bound and minimum lower bound over all subregions bound $f(x)$ from above and below on X. These bounds are valid over the set of points whose components satisfy S, since all such points belong to X by equation 7.8. DI splits X in each dimension i by dividing X_i into maximal intervals of the following types:

$$\begin{array}{ll} \textit{decreasing} & \text{SUP}(\frac{\partial f(x)}{\partial x_i}, \{\}) \leq 0 \\ \textit{increasing} & \text{INF}(\frac{\partial f(x)}{\partial x_i}, \{\}) \geq 0 \\ \textit{unknown} & \text{neither of the above.} \end{array} \quad (7.9)$$

If $f(x)$ increases in x_i on a subregion, it must attain its minimum over that subregion when x_i equals its left endpoint and its maximum when x_i equals its right endpoint. If $f(x)$ decreases, the endpoints are reversed. Either way, DI can collapse the subregion to a point in dimension i. Figure 7.8 shows the results of this procedure for the function $x_1^2 - x_2^2$ on the region $([-1,1],[-1,1])$. The upper and lower bounds, u_i and l_i, are found directly for each interval because no intervals are *unknown*.

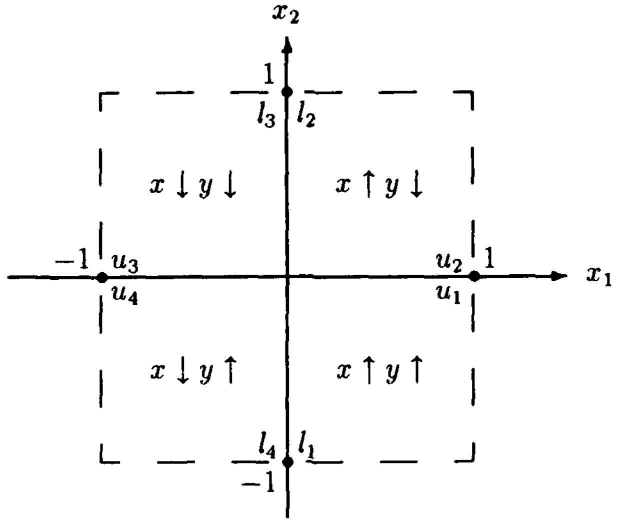


Figure 7.8: Derivative inspection for $x_1^2 - x_2^2$

Derivative inspection takes time proportional to the number of regions into which f's domain splits. For this reason, it only applies to functions whose partial derivatives all have finitely many zeroes in X. When every X_i consists solely of *increasing* and *decreasing* intervals, derivative inspection yields optimal bounds directly, since all regions reduce to points. Otherwise, one must use another bounding algorithm to calculate bounds on the non-trivial subregions. This two-step approach generally yields tighter bounds than applying the second algorithm directly on f's entire domain, since the subregions are smaller and often reduce to points along some dimensions.

7.3.4 Iterative Approximation

Iterative approximation, like derivative inspection, reduces the errors in bounds propagation and substitution caused by multiple occurrences of variables. Instead of bounding a function over its entire range directly, it subdivides the regions under consideration and combines the results. Intuitively, BP's choice of multiple worst case values for a variable causes less damage on smaller regions because all these values are less far apart. Figure 7.9 illustrates this idea for the function $x^2 - x$ on the interval $[0,1]$. Part (a) demonstrates that BP derives an overly pessimistic lower bound on $[0,1]$ because it minimizes both $-x$ and x^2 independently. Part (b) shows that this factor is less significant on smaller intervals: the minimum of the two lower bounds, $-3/4$, is a tighter bound for $x^2 - x$ on $[0,1]$ than that of part (a). One can obtain arbitrarily tight bounds by constructing sufficiently fine partitions.

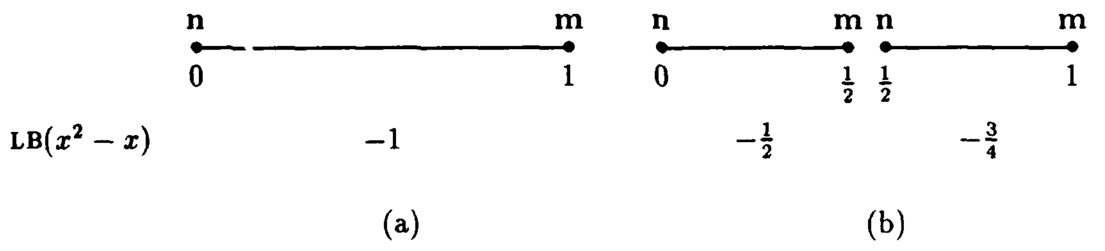


Figure 7.9: Illustration of iterative approximation for $x^2 - x$ on $[0,1]$. The symbols m and n mark the values of x that minimize $-x$ and x^2 respectively.

Iterative approximation generalizes interval subdivision to multivariate functions and increases its efficiency, using ideas from Moore [32] and Asaithambi et at. [2]. It converges

```
1. Apply DI to $f$ and $S$, pair each resulting region with its MUB value, sort the
   pairs in decreasing order of MUB, and set list to the sorted list.
2. $\langle Z, b\rangle \leftarrow head(list); list \leftarrow tail(list)$   {The MUB of $b$ is $Z$.}
3. If $U(X) = \emptyset$ or $b - \text{MLB}(Z) \leq \epsilon$ return $b$.
4. Choose an $i$ in $U(Z)$ which maximizes $w(Z_i)$.
5. Calculate $Z_1$ and $Z_2$ by bisecting $Z$ in dimension $i$ and collapsing the
   results in increasing and decreasing directions.
6. For $j = 1, 2$ do
   6.1 Delete from list every pair $\langle Z', b'\rangle$ for which $b' < f(m(Z_j))$.
   6.2 Insert $\langle Z_j, \text{MUB}(Z_j)\rangle$ into list in proper order.
7. Go to step 2.
```

Figure 7.10: Algorithm IA$(f(x), S, \epsilon)$

to the true bounds of any continuously differentiable function on a bounded domain. Let X_0 denote the range of the vector x in constraint set S, as defined in the previous section. The IA algorithm, shown in Figure 7.10, calculates an upper bound for a continuously differentiable function, $f(x)$, on a finite region, X_0, by iteratively dividing and shrinking X_0. It derives a lower bound by negating the upper bound of $-f(x)$. By Theorem 3, these bounds are valid over the set of points whose components satisfy S.

Let c denote the collection of subsets of X_0 produced by DI on input $f(x)$ and S. The least upper bound, LUB, of f on X_0 equals the maximum LUB of f over c, as explained in the previous section. IA uses a modified version of the UB function, MUB, to estimate the LUB on regions. Initially, it pairs each member of c with its MUB value and sorts the pairs in decreasing order of MUB. (The list can be implemented efficiently by a balanced tree.) The first MUB value in the list is an upper bound for f on X_0. At each iteration, IA splits the first region, Z, by bisecting an *unknown* component (defined in equation (7.9)) and inserting the resulting subregions into the list in proper order.

It would be inefficient to split Z in an *increasing* or *decreasing* direction, i, since the optimal value of x_i is known. In fact, the range of x_i consists of its optimal value. Derivative inspection imposes this condition on the initial partition and IA maintains it by collapsing each subregion of Z to its left endpoint in every *decreasing* direction and to its right endpoint in every *increasing* direction. Also, it would be pointless to consider regions on which f cannot attain its maximum. One such case occurs when the MUB value, b', of an entry $\langle Z', b'\rangle$ is less than f's value at the middle of Z_1 or Z_2. To improve efficiency, IA deletes these entries.

The MUB function is defined as follows:

$$\text{MUB}(X) = f(m(X)) + \text{UB}_X\left(\sum_{i\in U(X)}(x_i - m_i(X))\frac{\partial f(x)}{\partial x_i}\right) \tag{7.10}$$

The set $U(X)$ contains all dimensions in which $f(x)$ has *unknown* direction in X and the vector $m(X)$ denotes the middle of X, whose component $m_i(X)$ equals the midpoint of X_i. The MLB function is defined analogously, with LB replacing UB in equation (7.10). Moore [32] proves that these functions bound f on all subsets of X_0, that is

$$\forall X \subseteq X_0 \forall x \in X.\text{MLB}(X) \leq f(x) \leq \text{MUB}(X) \tag{7.11}$$

This implies that the LUB of f on X lies between its MUB and MLB.

Define the width of the interval $[a,b]$ as $b-a$ and the width of the region X, $w(X)$, as the maximum over the widths of its components. Moore proves that

$$\forall X \subseteq X_0.\text{MUB}(X) - \text{MLB}(X) \leq Lw(X) \tag{7.12}$$

with L a constant. This result assists in the proof that IA terminates.

Lemma 4 *Let $x = (x_1,\ldots,x_n)$ have finite range X in S. For every positive ϵ and function $f(x)$ continuously differentiable on X,* IA *terminates within*

$$M = \left(\frac{2L}{\epsilon}\right)^n \prod_{i=1}^{n} w(X_i) \tag{7.13}$$

iterations, where L is the constant of equation (7.12).

Proof: First, suppose *list* contains only the region X after step 1. Each interval X_k can be decomposed by a sequence of bisections into at most

$$\frac{2Lw(X_k)}{\epsilon} \tag{7.14}$$

subintervals without splitting some interval narrower than ϵ/L. All told, X can be decomposed into at most M regions without splitting such an interval. By the choice of i in step 4, IA can only split an interval narrower than ϵ/L if the region Z is narrower than ϵ/L. This never happens because such a Z satisfies the second disjunct of step 3, terminating the iteration, by equation (7.12). Hence, at most M iterations can occur.

Next, suppose *list* contains regions $X^1, \ldots, X^k$ after step 1. Applying the argument above to each region shows that IA terminates within

$$\left(\frac{2L}{\epsilon}\right)^n \sum_{j=1}^{k} \prod_{i=1}^{n} w(X_i^j) \tag{7.15}$$

iterations. The jth addend in this equation equals the volume of X^j, whereas the product in equation (7.13) equals the volume of X. The sum cannot exceed the product because the X^j are pairwise disjoint subsets of X. ∎

The following theorem establishes the correctness of iterative approximation:

Theorem 5 *Let x have finite range in S. For every positive ϵ and continuously differentiable function $f(x)$, iterative approximation calculates a number, b, satisfying*

$$\forall x.\mathit{satisfies}(x, S) \Rightarrow 0 \leq b - f(x) \leq \epsilon \tag{7.16}$$

Proof: It follows from Lemma 4 that IA terminates at step 3. If it halts because $U(Z)$ is empty, b is the LUB of f on Z because Z reduces to a point. If IA halts because $b - \text{MLB}(Z)$ is less than ϵ, then b is at most ϵ greater than the LUB by equation (7.11). Either way, b approximates the LUB of f on Z, which equals the LUB on X, within ϵ. This implies condition (7.16) by equation (7.8). ∎

For some applications and functions, other termination tests perform better than step 3 of Figure 7.10. Asaithambi et al. [2] prove that the test

$$\max\{\text{MUB}(Z_1), \text{MUB}(Z_2)\} \leq b \tag{7.17}$$

generates optimal bounds for rational functions. If one only needs to establish that f is not greater than a particular bound, b_0, then $b \leq b_0$ is a sufficient condition for termination. This case arises when the inequality prover proves that $c \leq d$ by establishing that $c - d$ has an upper bound of 0.

7.4 Related Work

In this section, I discuss, in order of increasing generality, existing programs that derive bounds and prove inequalities. As one would expect, the broader the domain of functions and constraints, the slower the program. The first class of systems bounds linear functions subject to linear constraints. Valdés-Pérez [44] analyzes sets of *simple linear inequalities* of the form $x - y \geq n$ with x and y variables and n a number. He uses graph search to test their consistency in cv time for c constraints and v variables. Malik and Binford [29] and Bledsoe [3] check sets of general linear constraints for consistency and calculate bounds on linear functions over consistent sets of constraints. Both methods require exponential time in the worst case, although they often perform better in practice. The former uses the Simplex algorithm, whereas the latter introduces preliminary versions of BOUNDER's substitution algorithms. Bledsoe defines SUP, SUPP, INF, and INFF for linear functions and constraints and proves the linear versions of Lemma 2 and Theorem 3. In fact, these algorithms produce *exact* bounds, as Shostak [40] proves.

The next class of systems bounds nonlinear functions, but allows only range constraints. All resemble BOUNDER's bounds propagation and all stem from Moore's [32] *interval arithmetic*. Moore introduces the rules for bounding elementary functions on finite domains by combining the bounds of their constituents. His algorithm takes linear time in the length of its input. Bundy [11] implements an *interval package* that resembles BP closely. It generalizes the combination rules of interval arithmetic to any function that has a finite number of extrema. If the user specifies the sign of a function's derivative over its domain, Bundy's program can perform interval arithmetic on it. Unlike BOUNDER's derivative inspection algorithm, it cannot derive this information for itself. Many other implementations of interval arithmetic exist, some in hardware.

Moore proposes a simple form of iterative approximation, which Skelboe [42], Asaithambi et al. [2], and Ratschek and Rokne [35, ch. 4] improve. BOUNDER's iterative approximation algorithm draws on all these sources.

Simmons [41] handles functions and constraints containing numbers, variables, and the four arithmetic operators. He augments interval arithmetic with simple algebraic simplification and inequality information. For example, suppose x lies in the interval $[-1, 1]$. Simmons simplifies $x - x$ to 0, whereas interval arithmetic produces the range $[-2, 2]$. He also deduces that $x \leq z$ from the constraints $x \leq y$ and $y \leq z$ by finding a path from x to z in the graph of known inequalities. The algorithm is linear in the total number of constraints. Although more powerful than BOUNDER's bounds propagation, Simmons's program is weaker than substitution. For example, it cannot deduce that $x^2 \geq y^2$ from the constraints $x \geq y$ and $y \geq 0$.

Brooks [10, sec. 3] extends Bledsoe's SUP and INF to nonlinear functions and argues informally that Lemma 2 and Theorem 3 hold for his algorithms. This argument must be faulty because his version of $\text{SUP}_H(e, \{\})$ recurses infinitely when e equals $x + 1/x$ or $x + x^2$, for instance. Brooks's program only exploits constraints among the variables of sums $rx + B$ and of products $x^n B$ with r real, x a variable of known sign, B an expression free of x, and n an integer. In other cases, it adds or multiplies the bounds of constituents, as in steps 3.1, 4.1.1, 4.2.1, and 4.3 of BOUNDER's SUP (Figure 7.5). These overly restrictive conditions rule out legitimate substitutions that steps 3.2, 4.1.2, and 4.2.2 permit. For example, BOUNDER can deduce that $1/x - 1/y \geq 0$ from the constraints $y \geq x$ and $x \geq 1$, but Brooks's algorithm cannot. On some functions and non-empty sets H, his algorithm makes recursive calls with H empty. This produces needlessly loose bounds and sometimes causes an infinite recursion.

Bundy and Welham [12, sec. 4] derive upper bounds for a variable x from an inequality $L \leq R$ by reformulating it as $x \leq U$ with U free of x. If U contains a single variable, they try to find its global maximum, M, by inspecting the sign of its second derivative at the zeroes of its first derivative. When successful, they bound x from above with M. Lower bounds and strict inequalities are treated analogously. Bundy and Welham use a modified version of the PRESS equation solver [12, 13] to isolate x. As discussed in section 7.2, inequality manipulation depends on the signs of the expressions involved. When this information is required, they use Bundy's interval package to try to derive it. The complexity of this algorithm is unclear, since PRESS can apply its simplification rules repeatedly, possibly producing large intermediate expressions. BOUNDER contains both steps of Bundy and Welham's bounding algorithm: its context manager derives bounds on variables from constraints, while its derivative inspection algorithm generalizes theirs to multivariate functions. PRESS may be able to exploit some constraints that BOUNDER ignores because it contains a stronger equation solver than does BOUNDER.

Constraint propagation is a popular AI technique for bounding expressions. Given a network of constraints between variables (or, more generally, terms), propagation incrementally refines the set of possible values for each variable by repeatedly selecting and enforcing constraints. For example, consider the constraints $a + b = c$ and $a \geq b + 1$ and the initial ranges $a, b, c \in [1, 10]$. Enforcing $a + b = c$ yields the tighter ranges: $a \in [1, 9]$, $b \in [1, 9]$, and $c \in [2, 10]$. Next, enforcing $a \geq b + 1$ yields $a \in [2, 9]$, $b \in [1, 8]$, and $c \in [2, 10]$. Davis [16] reviews a wide range of constraint propagation algorithms, classifies them, and derives their computational complexity. He shows that interval propagation is intractable or undecidable for all but the simplest constraints: order relations and simple linear inequalities. The standard Waltz constraint propagation algorithm may fail to terminate for general linear constraints, even though a polynomial time algorithm exists.

The final class of systems consists of theorem provers for predicate calculus that treat inequalities specially. These systems focus on general theorem proving, rather than problem-solving. They handle more logical connectives than BOUNDER, including disjunction and existential quantification, but fewer functions, typically just addition. Bledsoe and Hines [5] derive a restricted form of resolution that contains a theory of dense linear orders without endpoints. Bledsoe et al. [6] prove this form of resolution complete. Finally, Bledsoe et al. [4] extend a natural deduction system with rules for inequalities. Although none of these authors discuss complexity, all their algorithms must be at least exponential.

7.5 Conclusions

Current inequality reasoners are weak, brittle, or inefficient because they treat all inputs uniformly. Interval arithmetic systems, such as Bundy's and Simmons's, run quickly, but generate exceedingly pessimistic bounds when dependencies exist among components of functions. These dependencies are caused by constraints among variables or multiple occurrences of a variable, as discussed in Section 7.3.1. The upper bound of $a - b$ given $a \leq b$ demonstrates the first type, while the lower bound of $x^2 + x$ given no constraints demonstrates the second. Each of the remaining systems is brittle because it takes only one type of dependency into account. Iterative approximation, suggested by Moore, and derivative inspection, performed in the univariate case by Bundy and Welham, address the second type of dependency, but ignore the first. Conversely, substitution, used (in a limited form) by Brooks and Simmons, exploits constraints among variables, while ignoring multiple occurrences of variables. All these systems are inefficient because they apply a complex algorithm to every input without trying a simple one first.

BOUNDER overcomes the limitations of current inequality reasoners with its hierarchical strategy. It resolves simple problems in linear time with bounds propagation, an interval arithmetic based algorithm. If this fails, it applies more powerful methods. It uses substitution to analyze dependencies among variables and derivative inspection and iterative approximation to analyze multiple occurrences of variables. Together, these techniques cover far more cases than any single-algorithm system. Yet unlike those systems, BOUNDER does not waste time applying overly powerful methods to simple problems.

Every stage of PLR uses BOUNDER. Inequalities determine the extrema of piecewise linear functions, the stability characteristics of linear systems, the links in transition graphs, and the applicability of rule-out constraints in case analysis. The preconditions for most propositions in theoretical analysis include inequalities. More generally, an inequality reasoner like BOUNDER should be an important component of future general-purpose symbolic algebra packages.

Chapter 8

Manipulating Parameterized Functions

PLR invokes the QMR mathematical reasoner to analyze parameterized functions during piecewise linearization and the QS sketcher to draw phase diagrams. Also, it can depict the local behavior of a piecewise linear system by analyzing the solutions to its linear equations with QMR and sketching the results with QS. QMR solves the differential equations with the familiar algorithm from the theory of linear systems: Laplace transform, partial fractions expansion, and inverse Laplace transform. It uses a symbolic algebra package for these tasks and to simplify expressions, solve algebraic equations, and calculate limits.[1] I developed QMR and QS as part of my S.M. research. This chapter outlines them. See references [37, 38] for details about QMR and reference [39] for details about QS.

8.1 QMR

QMR infers the *qualitative properties* of functions: signs of the first and second derivatives, discontinuities, singularities, and asymptotes, which it records in data structures called *Q-behaviors.* It handles a large class of parameterized functions on the reals, including *extended elementary functions:* polynomials and compositions of exponentials, logarithms, trigonometric functions, inverse trigonometric functions, absolute values, maxima, and minima. For example, the Q-behavior for the function $(x-a)^{-1}$ appears in Figure 8.1.

[1]One version uses MACSYMA [30] and another uses the dynamicist's workbench [1]. A REDUCE version is being developed.

attribute	value
$f(x)$	$(x-a)^{-1}$
sign of $f'(x)$	undefined at a; negative elsewhere
sign of $f''(x)$	negative on $(-\infty, a)$; undefined at a; positive on (a, ∞)
discontinuities	$f(a)$ undefined; $\lim_{x \to a^-} f(x) = -\infty$; $\lim_{x \to a^+} f(x) = \infty$
singularities	$f'(a)$ undefined; $\lim_{x \to a} f'(x) = -\infty$

Figure 8.1: The Q-behavior of $f(x) = (x-a)^{-1}$

QMR can either analyze primitive functions from first principles of calculus or match them against stored patterns. It analyzes compound functions by analyzing their constituents recursively and combining the results. For instance, it constructs the Q-behavior for $(x-a)^2$ by substituting $x-a$ for x in the Q-behavior for x^2 and constructs the Q-behavior for $x+e^x$ by combining those of x and e^x. Three combination algorithms suffice because each compound function must consist of a sum, product, or functional composition of its constituents. QMR may produce several alternate Q-behaviors, depending on algebraic relations between parameters. For example, composing e^x with ax produces a decreasing, constant, or increasing function, depending on a's sign. It can also specialize Q-behaviors by instantiating some or all of their parameters.

QMR invokes BOUNDER to resolve inequalities over sets of constraints. It constructs a Q-behavior for each possibility when BOUNDER pronounces an inequality ambiguous. This strategy certainly derives all possible Q-behaviors for any input, but produces spurious ones as well in cases where BOUNDER fails to prove valid inequalities. No program can produce exact Q-behaviors for all extended elementary functions [36]. Nevertheless, QMR handles many complicated functions, as demonstrated in this thesis and in the above references.

8.2 Qualitative Sketching

QS sketches families of parameterized real univariate functions. It can sketch any function that QMR can analyze and any Q-behavior whatsoever. For example, Figure 8.2 contains the sketch of the Q-behavior that QMR derives for $(x-a)^{-1}$ (shown in Figure 8.1). QS is useful because sketches provide better insight into global behavior than verbal or other descriptions. It handles parameterized functions with infinite domains and unbounded values, whereas conventional plotting programs require bounded numeric functions on finite domains. Also, QS explicitly recognizes discontinuities, singularities, asymptotes, and periodicity, whereas conventional plotters cannot.

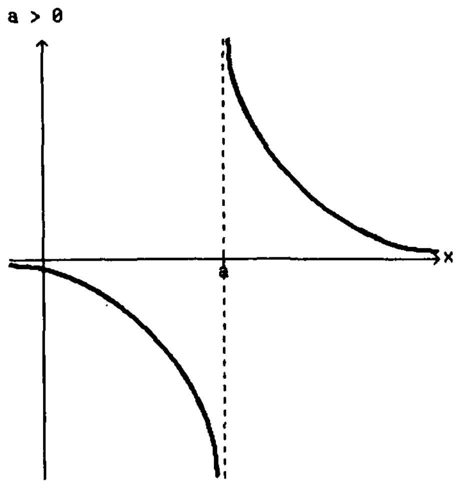


Figure 8.2: QS Sketch of $(x-a)^{-1}$

QS sketches a function by invoking QMR *to construct its* Q-behaviors and sketching each of them separately. It chooses a set of interesting points for each Q-behavior, picks an appropriate scale, lays out the points on the plane, and connects them with smooth curves. The interesting points for a Q-behavior consist of its boundaries, extrema, inflection points, discontinuities, singularities, and any additional points designated by the user. QMR can infer every type of interesting point (except for user designated ones) directly from Q-behaviors. The scale is a function from symbolic expressions to real numbers that preserves inequalities and bounds derivable by BOUNDER. This guarantees that the sketch expresses the key relations between interesting points. For example, the scaled value of a in Figure 8.2 should be positive and half that of $2a$.

Let p and q be two finite adjacent interesting points. If f is continuous at p and q, QS draws a smooth curve between the pairs $(p, f(p))$ and $(q, f(q))$ with initial derivative $f'(p)$ and final derivative $f'(q)$. If f is discontinuous and bounded at p or q, QS uses the left or right limit respectively in place of the (possibly undefined) value. Similarly, it substitutes the left and right derivatives for $f'(p)$ and $f'(q)$ at sharp points and cusps. An unbounded discontinuity at p is asymptotic to the line $x = p$. QS extends the sketch to the top of the y axis when the limit is ∞ and to the bottom when it is $-\infty$. It marks the asymptote with a dashed line. Infinite interesting points are treated analogously to unbounded finite ones. QS extends the sketch to the left end of the axis for $-\infty$ and the right end for ∞. If the limit is a finite number *lim*, it marks the asymptote $y = lim$ with a dashed line.

Figure 8.3 illustrates these ideas with two sketches. All constants are declared positive in these examples. The left-hand figure contains the sketch of $x^3 - 3ax$; its interesting points are: boundaries at $\pm\infty$, extrema at $\pm\sqrt{a}$, and an inflection point at 0. The right-hand figure contains the sketch of $\frac{x-a}{x-2a}$ with interesting points: $-\infty, 0, a, 2a$ and ∞. The sketch includes $(a, 0)$ and $(0, 1/2)$ because axis crossings have been designated interesting. There is a vertical asymptote at $x = 2a$ and a horizontal one at $y = 1$.

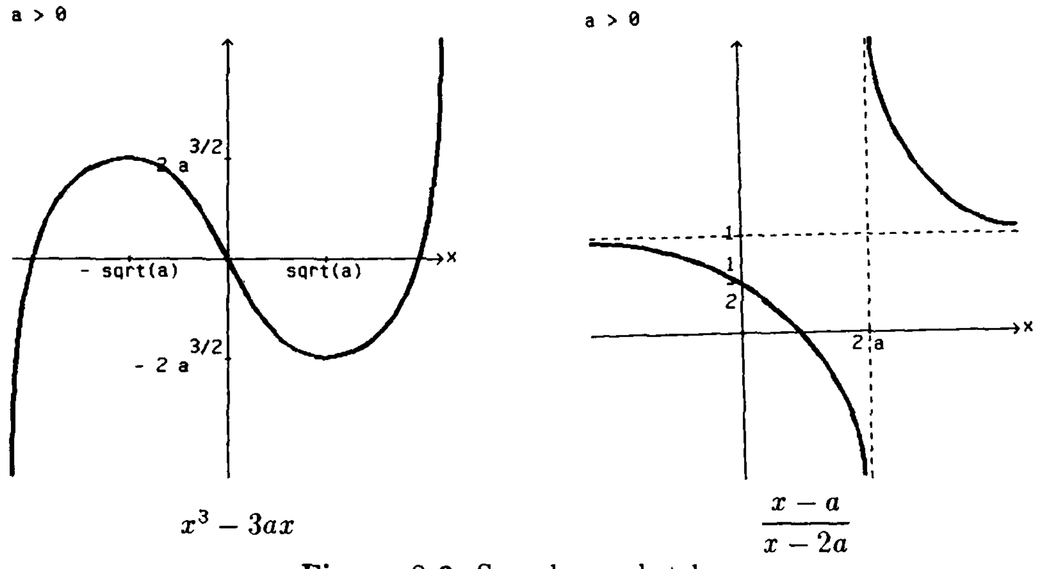


Figure 8.3: Sample QS sketches

8.3 Sketching Phase Diagrams

PLR sketches phase diagrams exactly as shown in the previous chapters, although I have substituted hand-drawn sketches to save paper and improve legibility. It invokes the QS sketcher to draw the axes and the boundaries between regions. It then labels the regions and marks the middle of each crossable boundary with an arrow. It can sketch systems containing parameterized boundaries because QS can handle parameterized functions. A sample screen image of PLR's graphics appears in Figure 8.4.

PLR *constructs individual trajectories of a system of differential equations through numeric simulation and superimposes them on its phase plane sketch.* It can simulate the original system or its piecewise linear approximations. The user can construct a complete phase diagram of a system or of its piecewise approximations by choosing several initial points in each region of the transition graph that PLR derives. He can assess the accuracy of a piecewise linearization by comparing its simulated trajectories with those of the original system. For example, Figure 8.4 compares PLR's phase diagram of the van der Pol equation (4.1) to that of the piecewise linear approximation (4.3) with $k, L, C = 1$. The approximation preserves the qualitative behavior of the original: all trajectories spiral toward a unique limit cycle. The numeric deviation between trajectories with identical initial conditions provides a quantitative measure of resemblance. It is small near the origin, but grows rapidly with $|x|$. *If the error grows unacceptably large, the user must refine the approximation*, as explained in Chapter 6.

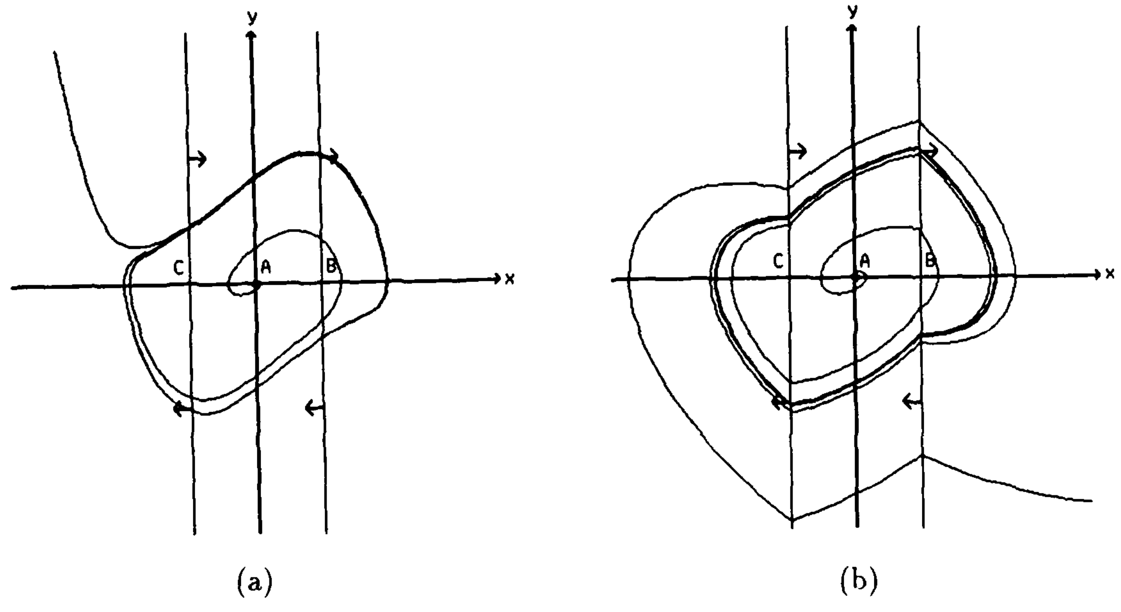


Figure 8.4: *Comparison of* PLR*'s phase diagrams of (a) the van der Pol equation (4.1) and (b) the piecewise linear approximation (4.3) with $k, L, C = 1$.*

Before simulating a system, PLR must instantiate its symbolic parameters with real numbers. Either the user provides the values, or PLR chooses defaults that satisfy the constraints specified for the parameters. The user enters an explicit initial value or chooses a point on the phase plane with the screen cursor. PLR constructs the trajectory through the selected point by simulating the system forward and backward in its free variable. The current implementation uses the Runge-Kutta method of order four [33, Chap. 15], but any other integration algorithm could be substituted transparently. I only describe the forward stage here; the other is analogous. PLR guides the simulation with the transition graph from piecewise analysis. It calculates the current region at each simulation step and records the sequence of previously traversed regions. It halts the simulation if this list grows longer than a prespecified multiple, k, of the number of regions in the phase plane. A trajectory that fulfills this condition must traverse a cycle in the transition graph k times. Heuristically, its global behavior becomes apparent after this number of traversals. PLR also halts if the simulation approaches a fixed point, becomes unbounded, or remains in one region forever.

Chapter 9

Review of Literature

Although the artificial intelligence community has ignored piecewise linear models, other modelers use them widely. Electrical engineers approximate nonlinear circuit components, such as transistors, with piecewise linear ones, as described by Chua [14] and Desoer and Kuh [18]. Kalman [23, 24] builds piecewise linear models of servo-mechanisms and other nonlinear devices through similar approximations. Itkis [20] and Zinober [45] develop tools for analyzing *variable structure systems:* feedback systems with piecewise linear control functions. Cook [15, Ch. 4] summarizes their methods and discusses other applications of piecewise linear differential equations.

Other commonly used techniques for analyzing nonlinear differential equations fall into the following categories: explicit solutions, dynamic systems theory, numeric simulation, and qualitative reasoning. Few systems have explicit solutions. Even when a modeler obtains an explicit solution, he must often expend considerable effort to infer its qualitative properties because of its complexity. Dynamic systems theory provides powerful tools for deriving qualitative behavior. At present, though, these tools apply to a limited number of systems and require a high level of mathematical sophistication of their users. PLR invokes dynamic systems theory when possible, as described in Chapters 5 and 6, but relies mainly on analysis of piecewise linear models.

Simulation yields low-level, numeric data about systems for specific parameter values and initial conditions. Modelers must interpret the data and generalize the results. This process becomes difficult for systems containing many parameters. Important properties of the model can be missed in interpretation for lack of raw data. For example, discontinuities and extrema might fall between the observed or simulated points, while asymptotes may exist beyond their range. One can also overlook important properties due to the sheer volume of raw data. Generalization can fail too, since a model need not behave in a certain manner for *all* parameter values just because it does so for certain ones.

Despite these problems, simulation handles most systems very well, including those too complex for other methods. PLR comes to augment simulation, not supplant it. In many applications, it tells practitioners everything they need to know about systems automatically, saving them the time and effort of simulation. In the remaining cases, it provides a basis for efficient simulation. One need only simulate in regions that contain potentially interesting, but unresolved, trajectories. The results of piecewise analysis also help guide the interpretation of simulation data.

Qualitative reasoning [7] (QR) is a paradigm for deriving the abstract behavior of dynamic systems.[1] It manipulates abstract versions of numbers, functions, and differential equations, called *qualitative values, quantities,* and *confluences,* respectively. Qualitative values correspond to points and intervals on the real line. They are organized into ordered sets of alternating intervals and points called *quantity spaces.* One useful quantity space, called IQ, consists of *minus:* the interval $(-\infty, 0)$, *zero:* the point 0, and *plus:* the interval $(0, \infty)$. I abbreviate these to $-$, 0, and $+$. A quantity is a function from an independent variable (usually time) to a quantity space; the nth *qualitative derivative* of a quantity q, denoted by $\partial^n q$, is the IQ value of its nth derivative. The QR versions of addition and multiplication are partial functions from pairs of qualitative values to qualitative values. For instance, the sum of the IQ values $+$ and $-$ is undefined, while their product equals $-$. The operator M^+ (M^-) denotes a smooth monotonically increasing (decreasing) transformation of its argument. Confluences state equalities between expressions: sums, products, and monotone transformations of quantities and their derivatives.

Instead of differential equations, QR uses sets of confluences to describe dynamic systems. Let us define the *state* of a quantity as its qualitative value and the IQ value of its first qualitative derivative. A system's state consists of the vector of the states of its quantities.

[1]The concepts and terminology of QR are not completely agreed on by practitioners. In the following discussion, I attempt to abstract the common themes of their research, while respecting common usage.

QR uses constraint propagation and properties of continuous functions to derive a graph of a system's possible behaviors from its confluences and the initial values of its quantities. The nodes represent system states and the links indicate possible transitions. Each walk through the graph describes a possible behavior of the system. Multiple walks result from ambiguities in the confluences and the analysis algorithm. For example, when b equals $+$ and c equals $-$ the confluence $a = b + c$ leaves the value of a ambiguous.

In its current form, QR falls far short of telling modelers what they need to know about nonlinear systems. It can only provide extremely abstract descriptions, such as "the quantity f increases for a while, reaches a maximum, and decreases thereafter." More information is required to analyze actual devices, physical laws, and other models: local properties in interesting regions such as estimates of maxima, minima, and rates of change as well as global properties such as stability, periodicity, limit cycles, and asymptotic behavior. QR abstracts away the details required to derive this information by representing dynamic systems with confluences instead of differential equations. It cannot even express many important functional properties, such as linearity, exponential decay, asymptotic approach, oscillation, damped oscillation, stability, and limit cycles.

QR also generates spurious behaviors. One cause, described by Kuipers [26], is the local character of its analysis. In addition, the abstract nature of confluences introduces ambiguities that differential equations preclude. For example, the equation $y' = y - y^2$ implies that y' is negative whenever y exceeds 1, whereas the corresponding confluence leaves the sign completely ambiguous. Consequently, QR concludes that y can increase toward infinity, even though it is bounded from above. Kuipers [25] notes that this type of ambiguity crops up in physiological models of second-order or higher. The same result holds for other domains.

Raiman [34] addresses a special case of this problem by incorporating assertions of the form "quantity a is negligible in relation to quantity b" into QR. His extension does not solve our example because neither y^2 nor y is negligible with respect to the other. Nor does similar work by Mavrovouniotis and Stephanopoulos [31].

From an abstract perspective, QR reduces to an extremely restricted form of piecewise linear reasoning.[2] The phase space for quantities $q_1, \ldots, q_n$ is the Cartesian product of their

[2]The reduction ignores the putative causal ordering that QR imposes during simulation. See Iwasaki and

quantity spaces $Q_1 \times \cdots \times Q_n$. Each Q_i divides phase space along the ith coordinate. Together they partition it into n-dimensional boxes. For example, the phase space for the IQ-valued quantities x and v (Figure 9.1) partitions into nine rectangles. Five of these are degenerate: four represent boundaries ([0, −], [0, +], [−, 0], and [+, 0]) and one represents a point ([0, 0]). I find degenerate regions confusing and unnecessary. It is better to represent boundaries and points explicitly, as does PLR.

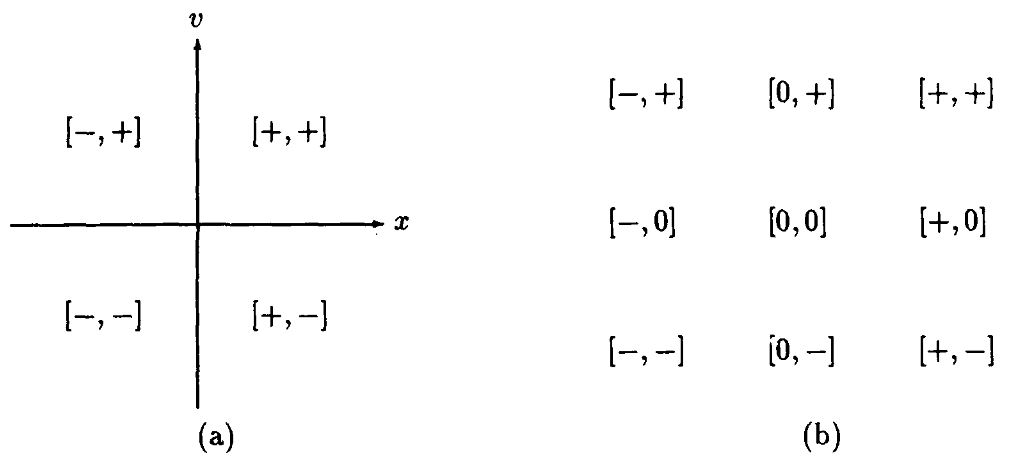


Figure 9.1: Phase Space for IQ-valued quantities x and v: (a) nondegenerate regions (b) all regions.

QR derives its graph of possible behaviors through a weak form of transition analysis. Let region A be adjacent to region B along the boundary $q_i = k$ with $A_i \leq k$ and $B_i \geq k$. QR inserts a link from A to B unless the confluences for A imply $\partial q_i \leq 0$ and from B to A unless the confluences for B imply $\partial q_i \geq 0$. The weakness of qualitative arithmetic, discussed above, causes this algorithm to produces more spurious links than does PLR's transition analysis.

Qualitative reasoning programs cannot perform case analysis. It is impossible to apply, or even express, PLR's rule-out constraints within the framework of confluences and qualitative arithmetic. Existing programs do not perform theoretical analysis either, although Lee [27] suggests doing so. He uses an ad hoc energy constraint to derive the asymptotic behavior of a *linear* spring, without exploiting the underlying general concept of a Liapunov function, discussed in Section 5.2.

We have seen that commonly used analysis techniques for nonlinear dynamic systems

Simon [21, 22] and de Kleer and Brown [17] for an alternate theory of causality.

are inadequate. Simulation yields low-level, numeric data about individual systems, whereas qualitative reasoning draws needlessly general conclusions because it ignores known analytic techniques. Some researchers are interested in QR as a tool for cognitive simulation, in which case its analysis may be appropriate. However, it is too weak a tool for expert-level reasoning about complex models. PLR provides an intermediate level of analysis that meets the needs of scientists and engineers. It remains to be seen whether one can match PLR's analysis by incorporating a richer set of confluences and stronger analysis algorithms into QR.

Chapter 10

Conclusions

This thesis explores automating the qualitative analysis of physical systems. Scientists and engineers model many physical systems with ordinary differential equations. They deduce the behavior of the systems by analyzing the equations. Automating the analysis is challenging because realistic models are nonlinear, hence difficult or impossible to solve explicitly. I have attacked the problem with a strategy whereby human experts analyze complicated differential equations. The resulting program, PLR, approximates intractable systems with tractable piecewise linear systems, analyzes the approximations, and draws *conclusions* about the originals. It augments the conclusions with theoretical analysis of global behavior when possible. The examples in Chapter 4 demonstrate the power of this approach.

10.1 Future Work

PLR is a first pass at exploiting piecewise linear approximations for automating the qualitative analysis of differential equations. There are many directions to extend each of its components. The current algorithm for making piecewise linear approximations fails on nonseparable systems. A smarter program would exploit analytic properties of systems, domain constraints, and user requirements to overcome that limitation. It could neglect semantically insignificant terms and linearize other terms on a case by case basis, rather than applying the current algorithm uniformly. When necessary, it would choose regions with nonlinear boundaries, thus trading off ease of analysis for expressive power. It could also approximate significant or rapidly-varying regions with more lines than other regions, as discussed in Chapter 6.

While constructing phase diagrams of piecewise linear systems, PLR could piece together global separatrices from critical local trajectories, such as asymptotes. For example, it could detect the separatrix of the undamped pendulum (Figure 1.4), which separates spinning trajectories from rocking ones, by linking the four asymptotes via the two critical trajectories, indicated by dotted curves. In the phase diagram of the Lienard equation (Figure 6.5b), it could link the descending asymptote with the dotted local trajectory, thus detecting the boundary between unbounded trajectories and ones that approach the origin. Case analysis would benefit from more subtle rule-out constraints for nonlinear boundaries, as discussed in Section 3.5.

PLR could determine the asymptotic behavior of additional systems if its current knowledge of dynamic systems theory, described in Chapter 5, were to be extended. In particular, it could exploit domain knowledge and analytic properties of systems to construct and modify Liapunov functions instead of relying solely on the prespecified energy function. Reasoning about the number, location, and stability properties of the limit cycles and separatrices in a system poses another challenge. The current implementation of PLR makes no use of the simulation data that it produces. The user must choose the parameter values and starting points, interpret the results, and vouch for their accuracy. Future versions of PLR should verify their approximations, transition graphs, and asymptotic predictions by performing and examining simulations of significant trajectories, such as separatrices and limit cycles.

10.1.1 Higher-Dimensional Systems

Although the current implementation of PLR only handles second-order systems, most of the algorithms extend to higher-order systems. The techniques for constructing and analyzing piecewise linear equations carry over directly from two dimensions to $k > 2$ dimensions. The linear regions generalize from rectangles to k-dimensional polygons; their boundaries generalize from curves to surfaces of dimension $k - 1$. The disposition of eigenvalues still determines local behavior, but the number of possibilities and asymptotes grows rapidly with k. The geometric criterion of transition analysis remains the same, as does its algebraic realization, $t \cdot n \leq 0$. The length of the inequality grows linearly in k. This increases the likelihood of BOUNDER introducing spurious transitions by failing to prove valid instances. However, the inequalities are linear, hence resolvable in polynomial time, when the boundaries between regions are hyperplanes. The rule out conditions R1–R3 of case analysis apply to k-dimensional systems. They grow harder to apply as k increases because larger values of k produce more complex local solutions.

Far fewer theoretical tools exist for determining the asymptotic behavior of k-dimensional systems than for two-dimensional ones. Liapunov functions (propositions 5.1, 5.2, and 5.3) and the divergence test (proposition 5.5) carry over, but the Poincaré-Bendixson theorem does not hold. Trajectories can wander chaotically through phase space without approaching a fixed-point, limit cycle, or separatrix. Sparrow [43] describes a parameterized system of three piecewise linear equations

$$\begin{cases} \dot{x}_1 = f(x_3) - x_1 \\ \dot{x}_2 = x_1 - x_2 \\ \dot{x}_3 = x_2 - x_3 \end{cases} \quad \text{with } f(x_3) = \begin{cases} -8.4x_3 + 3.35 & \text{for} \quad x_3 \le 3/7 \\ 8.4rx_3 - 0.25 - 3.6r & \text{for} \quad x_3 \ge 3/7 \end{cases} \tag{10.1}$$

that exhibits chaotic behavior for some values of r. His example is extremely simple; it contains a single piecewise linear term with two linear regions. Brockett [9] presents a more complicated example. Understanding chaotic behavior is an open research topic for dynamic systems researchers. The current state of the art precludes systematic analysis by humans, let alone by computers.

10.2 Scope and Limitations

PLR re' upon two hypotheses about piecewise linear systems: (1) they can reproduce the essential properties of nonlinear differential equations and (2) their global properties are derivable from local solutions and transition graphs. When the second hypothesis fails, PLR can turn to other methods. For example, it deduces by theoretical means that all the trajectories of the damped pendulum equation approach the origin, since piecewise analysis (Figures 4.7 and 4.8) leaves the question open. It could also extract such information from closer analysis of the piecewise equations, as described by Cook [15], or from simulation.

When the first hypothesis fails, piecewise analysis fails. For example, the first-order equation $y' = y^2$ with the initial condition $y(0) = y_0$ has the solution

$$y(t) = \frac{y_0}{1 - y_0 t} \tag{10.2}$$

which approaches $\pm\infty$ as t approaches $1/y_0$ and is undefined for $t = 1/y_0$. Piecewise linear equations cannot reproduce this behavior because their solutions, $ke^{\alpha t}$, are finite for all values of t. None of the examples that I have examined display this behavior. Perhaps physical constraints, such as the finite size and energy of real systems, prevent it from arising in practice. In any case, other types of essentially nonlinear behavior exist in nature. New tools are required to represent and analyze them.

Even when both hypotheses hold, PLR can fail due to limitations in its approximation algorithm, inequality prover, or other components. For example, it cannot construct the initial approximation of the horseshoe pendulum (Section 4.3) because QMR fails to derive the qualitative properties of the function g. Advances in inequality proving and computer algebra should solve many of these problems. Nevertheless, the current implementation of PLR handles many complicated systems. It can answer all of the questions some of the time, some of the questions all of the time, but not all of the questions all of the time. That's research.

In this thesis, I demonstrate that a computer program can automatically derive the qualitative behavior of complicated systems of ordinary differential equations by constructing and analyzing appropriate piecewise linear approximations. The program emulates the initial strategy of human experts. The next question is how to proceed when that strategy fails. Humans turn to a combination of theoretical tools and numeric analysis. I propose developing programs that do the same.

Bibliography

[1] Harold Abelson and Gerald Jay Sussman. *The Dynamicist's workbench: I.* AIM 955, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, MA, 02139, May 1987.

[2] N. S. Asaithambi, Shen Zuhe, and R. E. Moore. On computing the range of values. *Computing*, 28:225–237, 1982.

[3] W. W. Bledsoe. A new method for proving certain Presburger formulas. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 15–21, 1975.

[4] W. W. Bledsoe, Peter Bruell, and Robert Shostak. A prover for general inequalities. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 66–69, 1979.

[5] W. W. Bledsoe and Larry M. Hines. Variable elimination and chaining in a resolution-based prover for inequalities. In *Proceeding of the fifth conference on automated deduction*, Springer-Verlag, Les Arcs, France, July 1980.

[6] W. W. Bledsoe, K. Kunen, and R. Shostak. *Completeness results for inequality provers.* ATP 65, University of Texas, 1983.

[7] Daniel G. Bobrow, editor. *Qualitative Reasoning about Physical Systems.* M. I. T. Press, 1985.

[8] Fred Brauer and John A. Nohel. *The Qualitative Theory of Ordinary Differential Equations.* W.A. Benjamin, Inc., New York, 1969.

[9] R. W. Brockett. On conditions leading to chaos in feedback systems. In *Proceedings 21st IEEE Conference on Decision and Control*, pages 932–936, IEEE Control Systems Society, 1982.

[10] Rodney A. Brooks. Symbolic reasoning among 3-d models and 2-d images. *Artificial Intelligence*, 17:285–348, 1981.

[11] Alan Bundy. A generalized interval package and its use for semantic checking. *ACM Transactions on Mathematical Software*, 10(4):397–409, December 1984.

[12] Alan Bundy and Bob Welham. *Using meta-level descriptions for selective application of multiple rewrite rules in algebraic manipulation.* D.A.I. Working Paper 55, University of Edinburgh, Depatment of Artificial Intelligence. May 1979.

[13] Alan Bundy and Bob Welham. Using meta-level descriptions for selective application of multiple rewrite rules in algebraic manipulation. *Artificial Intelligence*, 16(2):189–211, May 1981.

[14] Leon O. Chua. *Introduction to nonlinear network theory.* McGraw-Hill, New York, 1969.

[15] P. A. Cook. *Nonlinear Dynamical Systems.* Prentice Hall International, Englewood Cliffs, N.J., 1982.

[16] Ernest Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32:281–331, 1987.

[17] Johan de Kleer and John Seely Brown. Theories of causal ordering. *Artificial Intelligence*, 29:33–61, 1986.

[18] Charles A. Desoer and Ernest S. Kuh. *Basic Circuit Theory.* McGraw-Hill, New York, 1969.

[19] Morris W. Hirsch and Stephen Smale. *Differential Equations, Dynamical Systems, and Linear Algebra.* Academic Press College Division, Orlando, Florida, 1974.

[20] U. Itkis. *Control systems of Variable Structure.* John Wiley, 1976.

[21] Yumi Iwasaki and Herbert A. Simon. Causality in device behavior. *Artificial Intelligence*, 29:3–32, 1986.

[22] Yumi Iwasaki and Herbert A. Simon. Theories of causal ordering: Reply to de Kleer and Brown. *Artificial Intelligence*, 29:63–72, 1986.

[23] R. E. Kalman. Phase-plane analysis of automatic control systems with nonlinear gain elements. *Transactions of the AIEE*, 73(2):383–390, 1954.

[24] R. E. Kalman. Physical and mathematical mechanisms of instability in nonlinear automatic control systems. *Transactions of the ASME*, 79:553–566, April 1957.

[25] Benjamin Kuipers. *Qualitative Simulation in Medical Physiology: A Progress Report.* TM 280, Massachussetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA, 02139, June 1985.

[26] Benjamin J. Kuipers. The limits of qualitative simulation. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 128–136, August 1985.

[27] W. W. Lee, C. Chiu, and B. J. Kuipers. Developments towards constraining qualitative simulation. In *Qualitative Physics Workshop Abstracts*, May 1987.

[28] Solomon Lefschetz. *Differential Equations: Geometric Theory.* John Wiley and Sons, New York, 1962.

[29] J. Malik and T. Binford. Reasoning in time and space. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 343–345, August 1983.

[30] Mathlab Group. *Macsyma Reference Manual.* MIT Laboratory for Computer Science, Cambridge, Mass., version 10 edition, January 1983.

[31] Michael Mavrovouniotis and George Stephanopoulos. Reasoning with orders of magnitude and approximate relations. In *Proceedings of the National Conference on Artificial Intelligence*, pages 626–630, American Association for Artificial Intelligence, 1987.

[32] Ramon E. Moore. *Methods and Applications of Interval Analysis. SIAM Studies in Applied Mathematics*, SIAM, Philadelphia, 1979.

[33] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes.* Cambridge University Press, Cambridge, England, 1987.

[34] Olivier Raiman. Order of magnitude reasoning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 100–104, American Association for Artificial Intelligence, 1986.

[35] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions.* Halsted Press: a division of John Wiley and Sons, New York, 1984.

[36] D. Richardson. Some unsolvable problems involving elementary functions of a real variable. *Journal of Symbolic Logic*, 33:511–520, 1968.

[37] Elisha P. Sacks. *Qualitative mathematical reasoning.* TR 329, Massachussetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA, 02139, 1985.

[38] Elisha P. Sacks. Qualitative mathematical reasoning. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 137–139, 1985.

[39] Elisha P. Sacks. Qualitative sketching of parameterized functions. In D. Sriram and R. A. Adey, editors, *Knowledge Based Expert Systems for Engineering: Classification, Education and Control*, pages 1–13, Computational Mechanics Publications, Boston, 1987.

[40] Robert E. Shostak. On the SUP-INF method for proving Presburger formulas. *Journal of the ACM*, 24:529–543, 1977.

[41] Reid Simmons. "Commonsense" arithmetic reasoning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 118–124, American Association for Artificial Intelligence, August 1986.

[42] S. Skelboe. Computation of rational functions. *BIT*, 14:87–95, 1974.

[43] C. T. Sparrow. Chaos in a three-dimensional single loop feedback system with a piecewise linear feedback function. *Journal of Mathematical Analysis and Applications*, 83(1):275–291, 1981.

[44] Raúl Valdés-Pérez. *Spatio-temporal reasoning and linear inequalities.* AIM 875, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, May 1986.

[45] A. S. I. Zinober. Controller design using the theory of variable structure systems. In C. J. Harris and S. A. Billings, editors, *Self-tuning and Adaptive Control*, chapter 9, pages 204–229, Peter Peregrinus LTD, 1981.